Clube Delphi

Ano 13 :: Edição 155 :: R$ 14,90

DEVMEDIA

FastReport do básico ao avançado
Relatórios com gráficos e cross-tab

Sua primeira aplicação Android no Delphi XE 5
Prepare o ambiente de trabalho e comece já!

Novidades do Delphi XE5

O que há de novo na Plataforma, Compilador e muito mais

Prepare-se para o Firebird 3
Conheça as novidades da versão Alpha

Introdução ao UniDAC
Explore mais uma opção para acesso a dados

ISSN 1517990-7

DEVMEDIA

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior portal para desenvolvedores da América Latina!

20 mil posts

430 mil cadastrados

10 milhões de page-views por mês

http://www.devmedia.com.br/

DEVMEDIA

Clube Delphi

Ano 13 - 155ª Edição 2014 ISSN 1517990-7 Impresso no Brasil

Editor Geral

Paulo Quicoli (pauloquicoli@gmail.com)

Equipe Editorial

Guinther Pauli (guintherpauli@gmail.com)

Fabrício Hissao Kawata (fabricio.kawata@bol.com.br)

Giuliano Scombatti Pinto (giuliano@sygnux.com.br)

Daniel Sobrinho Laporte (daniel.laporte@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Consultor Técnico

Wesley Yamazack (yamazack@gmail.com)

Capa e Diagramação

Romulo Araujo

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

Paulo Quicoli - **Editor Geral**

pauloquicoli@gmail.com
quicoli.wordpress.com
twitter.com/pauloquicoli

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DEVMEDIA

Sumário

Conteúdo sobre Novidades

04 – Novidades do Delphi XE5
[*Fabrício Hissao Kawata*]

Artigo no estilo Curso

13 – Introdução ao UniDAC – Parte 2
[*Fabrício Hissao Kawata*]

Artigo no estilo Solução Completa

22 – FastReport: Criando relatórios para VCL e FMX
[*Paulo Quicoli*]

Conteúdo sobre Novidades

32 – Novidades do Firebird 3
[*Paulo Quicoli*]

Artigo no estilo Solução Completa

40 – Desenvolvimento Android no Delphi
[*Fabrício Hissao Kawata*]

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Novidades do Delphi XE5

Conheça tudo sobre a nova versão do IDE

O termo mobilidade é a ordem da vez no estágio atual do mercado corporativo e isso se reflete diretamente no cenário do desenvolvimento de software. Cumprindo com sua premissa de se manter atualizado com as tendências correntes, o Delphi provê, através de sua mais recente versão (XE5), o suporte nativo para as duas principais plataformas móveis do cenário atual: Android e iOS. Tudo isso, é provido por um framework de desenvolvimento único (FireMonkey), a partir da mesma base de código e forma de programar habituais. Adicionalmente, as aplicações móveis produzidas tratam-se de soluções realmente nativas, livres de scripts e de qualquer outra tecnologia intermediária. Isso resulta num eficiente aproveitamento do uso de toda parte gráfica (GPU), de processamento (CPU), de serviços e sensores do dispositivo (hardware).

O presente artigo apresenta então um overview sobre as principais novidades desta nova versão do Delphi, cujo foco se dá essencialmente sobre seu novo suporte a Android. Até mesmo por este fato, a ferramenta vem sendo tratada em meio à comunidade, como "Delphi for Android". Logo, a intenção principal aqui é situar o leitor sobre o estágio atual do produto, pontuando os aspectos técnicos (na teoria e na prática) relacionados a cada novo recurso ou melhoria adicionada.

Fique por dentro

O foco do presente artigo é apresentar ao leitor, de forma geral, as principais novidades da mais nova versão do Delphi, a XE5. Já de antemão vale a pena mencionar que, de forma similar ao que ocorreu em seu lançamento anterior (XE4), o IDE apresenta como tema central o desenvolvimento multidispositivo e multiplataforma. Pesquisas atuais apontam o Android como o sistema operacional mais operante entre os dispositivos utilizados, tomando conta de uma grande parcela do mercado. De forma a se manter alinhado com esta tendência, o Delphi XE5 traz como grande atrativo seu novo suporte a Android. Isso significa dizer que a partir de uma mesma base de código, o desenvolvedor é capaz de construir aplicações que irão rodar em PCs, Tablets e Smartphones, sobre as principais plataformas do cenário atual do mundo tecnológico (Windows, Mac OS X, iOS e Android).

Android

O grande atrativo do Delphi XE5 é, sem sombra de dúvidas, o suporte ao desenvolvimento móvel para Android. Assim como as adições recentes à ferramenta, tal suporte culmina na introdução de novos conceitos e no provimento de novos recursos. Todavia, apesar de uma aparente dificuldade inicial esperada, a proposta da Embarcadero é a de manter a unicidade de todo o processo de desenvolvimento, seja ele voltado a Desktop ou Mobile, fazendo com que haja um reaproveitamento do conhecimento que o desenvolvedor Delphi adquiriu ao longo dos anos.

Novo compilador exclusivo

Dada sua constante evolução, até sua versão XE3, o Delphi trazia consigo três opções de compiladores, cada qual direcionado a uma plataforma específica: DCC32 para Windows 32-bits, DCC64 para Windows 64-bits e DCCOSX para Mac OS X. Posteriormente, com o lançamento do XE4 e ao então provimento do suporte nativo ao desenvolvimento móvel para iOS, este número aumentou para cinco. Dessa forma, dois novos compiladores foram adicionados ao contexto do produto, sendo DCCIOSARM para dispositivos iOS e DCCIOS32 para simuladores iOS. Agora no XE5, em razão da expansão do suporte nativo ao desenvolvimento móvel da ferramenta para a plataforma Android, um novo compilador exclusivo é adicionado ao leque existente. Nominalmente, ele é tido como DCCAARM e apresentado ao grande público como Delphi Android Compiler. Dessa forma, a partir de seu uso é que se torna possível a compilação de aplicações Delphi para emuladores e dispositivos Android.

Em vista destes novos "termos" recentemente surgidos, com relação aos compiladores que acompanham o produto, cabem alguns comentários. Inicialmente pelo fato do Delphi ter sido concebido como uma ferramenta de desenvolvimento para Windows, os compiladores para esta plataforma sempre se enquadraram num grupo conhecido como Delphi Compilers. Acompanhando sua evolução, todos os recentes compiladores incorporados à ferramenta, e que fundamentalmente são direcionados a outras plataformas (Mac OS X, iOS e, agora, Android), dão origem a uma outra segmentação que resulta num novo grupo denominado Delphi Cross Compilers. Ainda seguindo por esta linha, em se tratando especificamente das plataformas móveis suportadas (iOS e Android), os respectivos compiladores se enquadram num subgrupo nomeado como Delphi Mobile Compilers.

Versões e dispositivos suportados

Em termos gerais os dispositivos físicos (smartphones e tablets) Android suportados atualmente pelo Delphi XE5 são aqueles providos de processador ARMv7. Essa garantia é fruto dos resultados de uma série de testes pré-release efetuados pela equipe de desenvolvimento da Embarcadero, sobre diversos dispositivos reais. Tais testes envolvem basicamente a implantação e execução de aplicações FireMonkey Mobile nos dispositivos abordados. Algumas das aplicações acompanham a própria instalação do IDE, em sua pasta Samples (Ex: PhotoEditorDemo, SensorInfo e FlashLight), se tornando úteis para testes iniciais e pessoais. Uma lista completa dos aparelhos testados e os respectivos percentuais de sucesso são disponibilizados no Help online do produto. Este percentual é então provido como forma de garantir o máximo de transparência à comunidade, prevendo eventuais contratempos. HTC One X, HTC Desire HD e Kindle Fire HD são alguns dos dispositivos, cujos testes não atingiram o valor máximo esperado. Em complemento, apenas como demonstrativo, a seguir são expostos alguns dos dispositivos mais populares suportados com 100% de aceitação nos testes:

- Nexus (4, 7, 7 HD, 10);
- Galaxy (SII, SIII e S4);
- Galaxy Tab2 7;
- LG Optimus L5;
- Orange Nivo.

Como já é de conhecimento da grande maioria, o Android, como qualquer outro sistema operacional, contempla uma série de versões de software. Assim como é de praxe no cenário atual, cada versão estável é batizada com um nome representativo, uma espécie de apelido. No Delphi XE5, o suporte previsto inicial abrange três das principais versões do SO: Jelly Bean (Android 4.1.x, 4.2.x, e 4.3.x), Ice Cream Sandwich (Android 4.0.3 e 4.0.4) e Gingerbread (Android 2.3.3 – 2.3.7). Essa especificação é criteriosamente apresentada ao desenvolvedor, em razão do não previsto suporte a algumas outras versões do Android, tal qual a 3.x (Honeycomb).

Requisitos de desenvolvimento para Android

Nesta nova versão do IDE, sua própria instalação já provê todas as ferramentas necessárias para o desenvolvimento de aplicações para a plataforma Android, de forma a atender a todas as etapas do processo. Num exemplo claro, por meio do ambiente de desenvolvimento torna-se possível a execução da aplicação em um emulador e o posterior deploy para um dispositivo físico. Obviamente todo este contexto está relacionado a uma série requisitos essenciais para a efetiva conclusão de cada etapa envolvida. Tomando como base o suporte ao desenvolvimento móvel existente no IDE até então, para iOS, tais requisitos acabavam por formar uma espécie de check-list:

1. Tornar-se membro do Programa oficial de desenvolvedores da Apple, para iOS (iOS Developer Program). Este requisito é essencial para o deploy e distribuição das aplicações para a plataforma em dispositivos físicos e na loja oficial de aplicativos da Apple (Apple Store);
2. Obter um certificado de desenvolvedor (Developer Certificate), uma espécie de assinatura digital que associa o desenvolvedor e sua identidade digital com suas aplicações para a plataforma;
3. Determinar um perfil de provisionamento (Provisioning Profile), para que o desenvolvedor seja capaz de executar suas aplicações para iOS em um determinado dispositivo, distribuí-la de forma privada (ad hoc), ou mesmo torná-la candidata à Apple Store;
4. Criar perfil de conexão (Connection Profile), relacionado ao suporte previsto tanto a um dispositivo físico quanto a um simulador iOS;
5. Configuração de um ambiente de desenvolvimento Mac, paralelo ao ambiente ao qual o Delphi está inserido, para a efetiva execução das aplicações;
6. Conexão de um dispositivo iOS (iPhone, iPod Touch e iPad) via USB, para deploy no aparelho;
7. Instalação do conjunto de ferramentas de desenvolvimento (SDK – Software Development Kit) condizente (iOS Device SDK), que irá fornecer um conjunto de arquivos para a criação de aplicativos para a plataforma em questão.

Em contraposto a este cenário, o contexto do novo suporte a Android é mais enxuto, o que acaba por simplificar todo o processo. Como exemplo maior está ausência da necessidade de se configurar um ambiente de desenvolvimento paralelo. Por conseguinte, os requisitos essenciais desta abordagem podem ser pontuados, basicamente, pela instalação de três elementos: o Android NDK (ver **BOX 1**), o Android SDK condizente com a versão utilizada do Android, e o driver USB relativo ao dispositivo físico envolvido. Por padrão, no XE5, os dois primeiros já são providos ao desenvolvedor por meio da própria instalação do Delphi, conforme mostra a **Figura 1**. Em complemento pode-se ainda incluir a este cenário a criação de um arquivo específico, denominado keystore file para uma eventual submissão das aplicações construídas para as lojas de aplicativos Android, tal qual a Google Play.

BOX 1. Android NDK

No contexto Android, NDK é o acrônimo para Native Development Kit, termo este bastante utilizado no desenvolvimento de aplicativos para a plataforma. De forma prática, o Android NDK pode ser definido como sendo um conjunto de ferramentas que possibilita a implementação de parte do aplicativo, fazendo uso de linguagens de código nativo, tais como C e C++. Por esta razão, nem todos os aplicativos desenvolvidos necessariamente precisam do Android NDK, além do que seu uso não resulta em melhoras substanciais, sob o aspecto de desempenho. Atualmente o Delphi XE5 faz uso interno de um pacote NDK (android-ndk-r8e) e que, por padrão, é instalado junto da ferramenta.

Conectividade a fontes de dados

Na realidade atual a disponibilidade de dados às aplicações Mobile estão relacionadas a diferentes fontes de informação. Neste aspecto, de forma prática, é possível caracterizar três elementos como sendo os principais no cenário recente do desenvolvimento de software: bancos de dados, servidores de aplicação e serviços

cloud (na nuvem). Transpondo esta explanação ao contexto do Delphi e, especialmente ao novo suporte a Android, os três elementos citados são então refletidos nos próprios recursos providos pela ferramenta: gama de SGBDs compatíveis, servidores DataSnap e suporte a serviços cloud da Amazon e Azure. Logo, para usufruir desta gama de possibilidades nas novas aplicações móveis (FireMonkey Mobile) para Android, o desenvolvedor conta com praticamente os mesmos recursos e, fundamentalmente, os mesmos componentes já tradicionalmente utilizados em projetos Desktop para Windows (VCL Forms). Sendo assim, para conectividade a bancos de dados, são três as opções de bibliotecas disponíveis para projetos FireMonkey Mobile no XE5: dbExpress, InterBase Express (IBX) e FireDAC. Já para a conectividade com DataSnap Servers, as aplicações contam, por exemplo, com os componentes da guia REST Client e DataSnap Client, presentes na Tool Palette. Em complemento, para a parte cloud, também estão disponíveis para este tipo de aplicação os componentes TAzureConnectionInfo e TAmazonConnectionInfo.

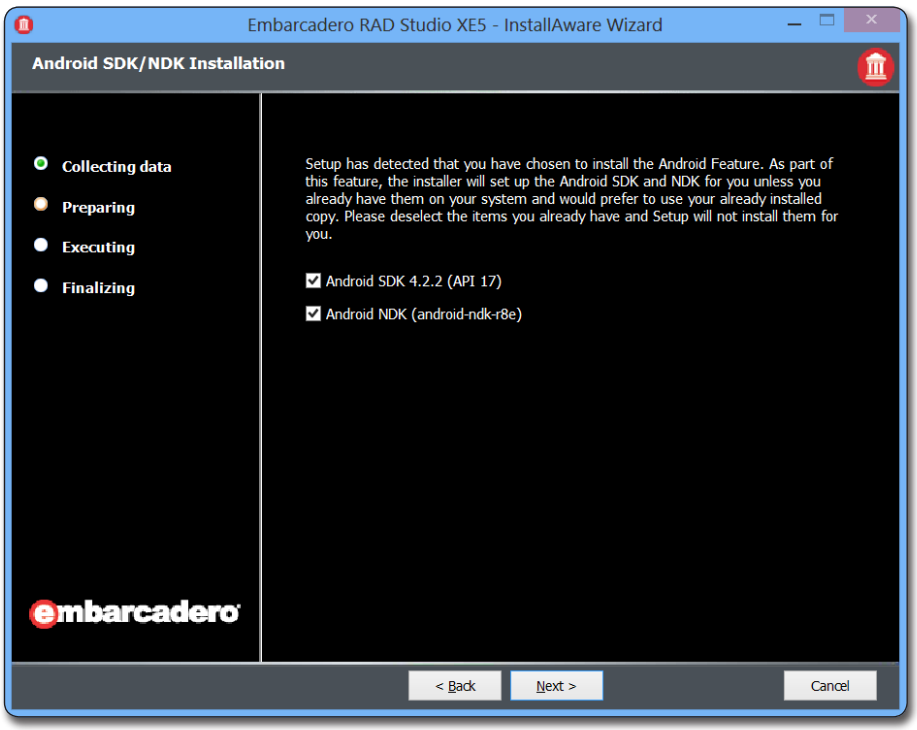

Figura 1. Instalação do Delphi XE5 – Android SDK e Android NDK

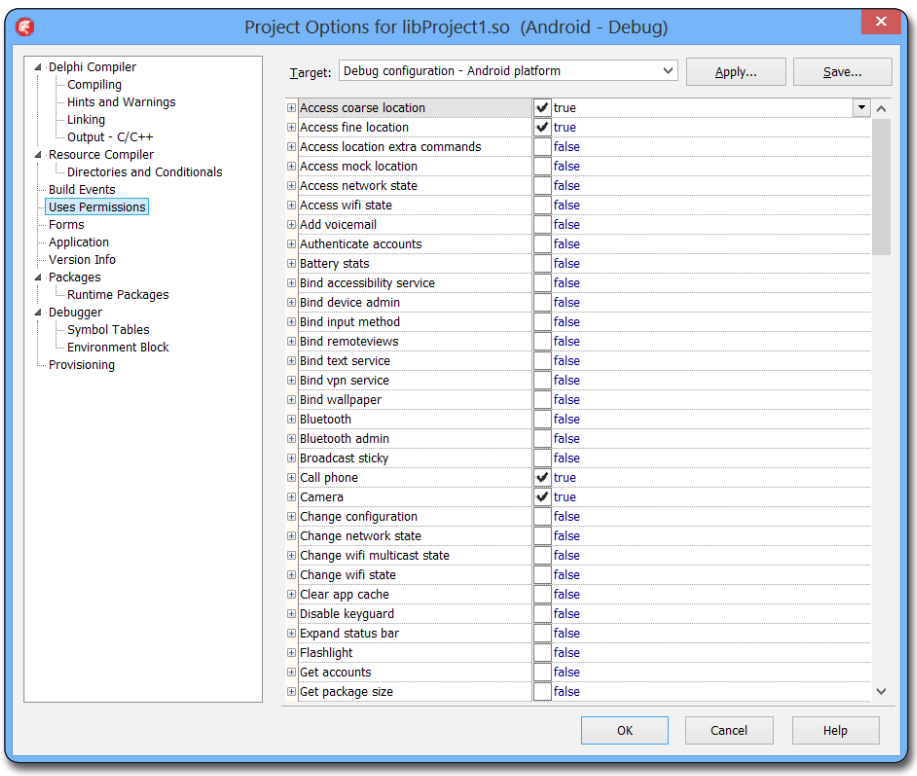

Figura 2. Opções de Projeto – User Permissions

Opções de projeto Android

No Delphi, as configurações de um determinado projeto ficam concentradas no que é conhecido como "Opções de projeto", disponível por meio do menu Project>Options (ou atalho Ctrl+Shift+F11) do IDE. Por meio delas é que são definidas questões específicas do projeto, tal como a autocriação de formulários, versionamento da aplicação, eventos pré e pós Build e pacotes de runtime. No XE5, especificamente para projetos Android (FireMonkey Mobile Application), novos detalhes são então acrescidos às opções de projeto.

O primeiro deles diz respeito a uma nova página, denominada User Permissions, que permite se especificar as permissões, referente a recursos do dispositivo que a aplicação irá requerer para operar no aparelho em que está sendo instalada. Sendo assim, ao acessar esta opção o desenvolvedor se depara com uma lista de itens "marcáveis" (checks), cada qual condizente com um recurso de dispositivo, conforme mostra a **Figura 2**. A seguir são exemplificados alguns dos itens disponíveis:

- Battery Stats: permite que o aplicativo colete informações sobre estatísticas da bateria do dispositivo;
- Bluetooth: permite que o aplicativo se conecte a dispositivos pareados via Bluetooth;
- Call Phone: possibilita a aplicação iniciar uma chamada telefônica pelo aparelho;

• Camera: acessibilidade à câmera do dispositivo;
• Change Configuration: possibilita que o aplicativo altere as configurações atuais do dispositivo;
• Flashligh: acesso à lanterna do dispositivo;
• Send SMS: possibilita à aplicação o envio de mensagens SMS pelo dispositivo;
• Write Calendar: permite que o aplicativo escreva dados para o calendário do usuário.

Apenas para mencionar, uma situação clara do uso de User Permissions se dá nos casos onde a aplicação Android que está sendo construída faça uso do componente FireMonkey de calendário, o TCalendarEdit. Neste cenário, a funcionalidade do próprio componente acaba por exigir as permissões relacionadas a calendário (Read Calendar e Write Calendar). Para o usuário final, este tipo de configuração se reflete em uma mensagem que é lhe mostrada no primeiro momento de uso da aplicação, onde é requisitada a confirmação de sua permissão para o uso dos recursos exigidos pela mesma. Em vista de todas as nuances apresentadas, uma dica já conhecida referente ao desenvolvimento Android é desabilitar toda e qualquer permissão que a aplicação não fará uso, tornando-a íntegra como também evitando possíveis transtornos ao usuário final.

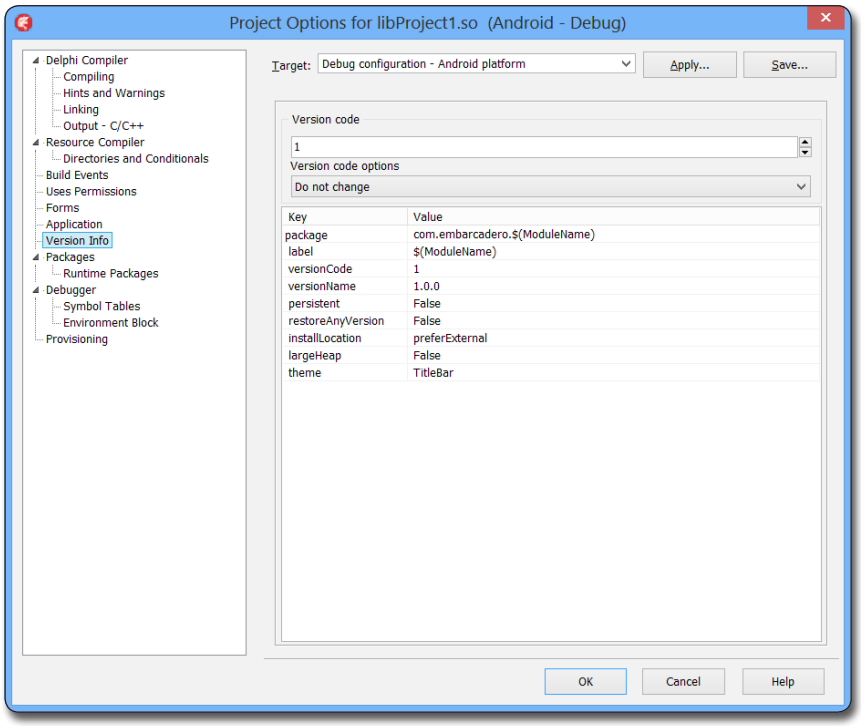

Figura 3. Opções de Projeto – Version Info

Ainda sobre as opções de configuração para projetos Android, outro novo detalhe relacionado é o item Version Info (**Figura 3**). Ao contrário de seu comportamento tradicional em projetos Win32, aqui Version Info provê a definição de determinadas informações a serem usadas pelo arquivo de projeto AndroidManifest.xml. Tais informações se apresentam sob a forma de campos editáveis, tais como:
• Version Code: campo que define o código (número em valor inteiro) da versão da aplicação na plataforma. Seu uso torna-se de fundamental importância no processo de distribuição da aplicação em lojas de aplicativos Android. Isto porque, lojas como o Google Play acabam por exigir que uma versão mais nova da aplicação tenha um código de versão superior à anteriormente disponibilizada;
• Key/Value: lista de itens chave-valor que definem algumas informações adicionais sobre a aplicação. Por padrão, esta lista traz algumas chaves (Key) definidas, tais como "label", que especifica um nome de exibição para aplicação, e "versionName", que determina uma representação em texto da versão corrente;

IDE

Como já é de praxe, a cada novo release o Delphi apresenta mudanças significativas ou superficiais em seu ambiente integrado de desenvolvimento (IDE). Nesta versão XE5, algumas destas são plenamente perceptíveis e estão relacionadas diretamente ao novo suporte ao desenvolvimento de aplicações móveis.

Target Platforms

A primeira novidade diz respeito ao gerenciador de projetos da ferramenta (Project Manager), sob o contexto de projetos do tipo FireMonkey Mobile Application. Mais especificamente o nó Target Platforms, que indica as plataformas de destino de um projeto, apresenta agora dois novos elementos filhos para a cada target platform relacionada. Logo, estes elementos são expostos com os nomes de Target e Configuration, tendo seus ícones em forma de pasta, tal como mostrado na **Figura 4**. Cada um deles exerce então um papel funcional diferente. Em Target são listados os dispositivos, emuladores e simuladores relacionados à plataforma alvo, que se encontram disponíveis ao projeto corrente. Já em Configuration são agrupadas e dispostas as configurações de build (build configurations) existentes, específicas à plataforma de destino (Ex: Debug, Application Store e Ad hoc para a target platform iOS Device). Adicionalmente, a visualização de tais elementos varia para os demais tipos de projetos. Num exemplo simples, um projeto tradicional do tipo VCL Forms Application não contempla nenhum dos dois elementos, enquanto que para um projeto FireMonkey Desktop Application, especificamente para a Target Platform OS X, somente o nó Configuration é disponibilizado.

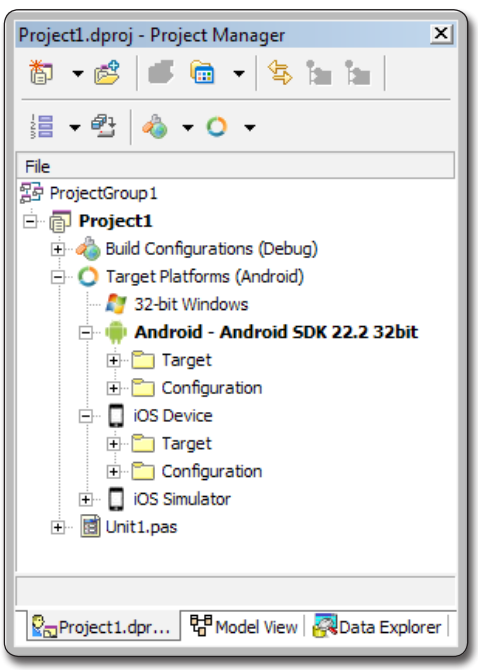

Figura 4. Target Platforms – Target e Configuration

Além desta, outra novidade já esperada e claramente perceptível é a disponibilização de uma nova Target Platform exclusiva para Android. Dessa forma, agora no Delphi XE5 todo projeto FireMonkey Mobile Application já contempla, por padrão, todas as Target Platforms relacionadas ao contexto móvel atual da ferramenta – iOS Device, iOS Simulator e Android – conforme pode ser visto na **Figura 4**. Conforme já mencionado anteriormente, a plataforma Android está presente em diversos dispositivos diferentes, incluindo distintas versões do sistema operacional. Por esta razão, a Target Platform para Android já é habilitada a listar, em seu nó Target, todos os eventuais dispositivos conectados ao ambiente de desenvolvimento, bem como os emuladores de cada versão utilizada.

Form Designer

O desenvolvimento móvel com o Delphi é provido por meio das aplicações FireMonkey Mobile. Em projetos deste tipo, o designer de formulário do IDE, tradicionalmente conhecido como Form Designer, acaba por ganhar outra denominação, sendo tratado como FireMonkey Mobile Form Designer. Apesar disso sua funcionalidade principal permanece a mesma, que é a de auxiliar o desenvolvedor na definição efetiva do design do(s) formulário(s) de suas aplicações. Ainda sob o contexto das aplicações móveis está a utilização do termo Design Device, que está relacionado ao aspecto especial dado ao formulário, em tempo de design, que esboça sua aparência final em um determinado dispositivo específico. Até a versão XE4 as opções de Design Devices disponíveis se resumiam à família de dispositivos Apple (iPhone e iPad), todavia agora na versão XE5 esta gama aumenta, em virtude do suporte a Android e a variedade de dispositivos contemplados por esta plataforma. Google Nexus 4, Google Nexus 7, Samsung Galaxy S2 e Samsung Galaxy S4 são alguns dos novos Design Devices disponibilizados nativamente pelo IDE, conforme mostra a **Figura 5**.

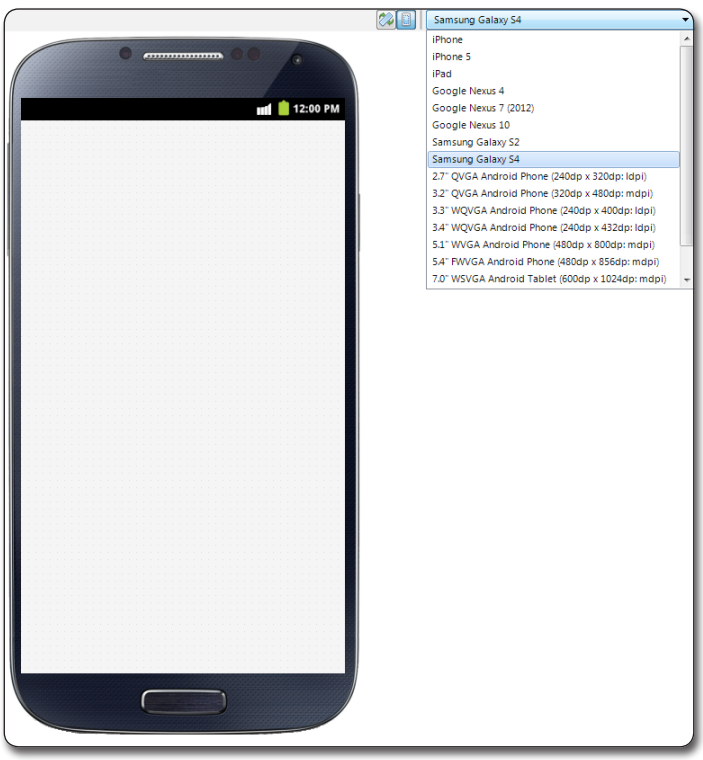

Figura 5. FireMonkey Mobile Designer – Design Device

Um detalhe importante a ser mencionado sobre Design Devices é que sua função no processo de desenvolvimento é, única e exclusivamente, a de servir como um guia visual ao formulário

que está sendo desenhado. Isso significa dizer que o uso de determinado Design Device não incidirá, de forma direta, nas configurações do Form, por exemplo, sobre suas dimensões. Por conseguinte, ao fazer o deploy de uma aplicação FireMonkey Mobile, por padrão, já há a garantia de que seus formulários serão ajustados automaticamente para o dispositivo em questão, independente do Design Device utilizado durante o desenvolvimento.

Device Manager

Em razão do pleno suporte ao desenvolvimento móvel incorporado ao Delphi, agora no XE5 um novo item é adicionado à seção de configurações do ambiente (Environment), localizado nas opções da ferramenta (Tools>Options). Nomeado como Device Manager, o novo nó fica associado às opções do Form Designer (**Figura 6**), uma vez que envolve a definição dos designs dos dispositivos que estarão disponíveis no Designer de formulários de projetos do tipo FireMonkey Mobile Application. Desta maneira, este é o local onde o desenvolvedor poderá definir a configuração de um novo dispositivo, editar as configurações de um dispositivo existente, ou mesmo excluir aqueles que estão fora do contexto de seus projetos. Adicionalmente, é possível ainda, mediante a lista de dispositivos apresentados, definir aquele que será a opção padrão para os projetos de aplicações Mobile. Como não poderia deixar de ser, neste release inicial do Delphi XE5, em vista de seu novo e aclamado suporte a Android, o dispositivo marcado como "padrão" está relacionado à plataforma, sendo o Google Nexus 4.

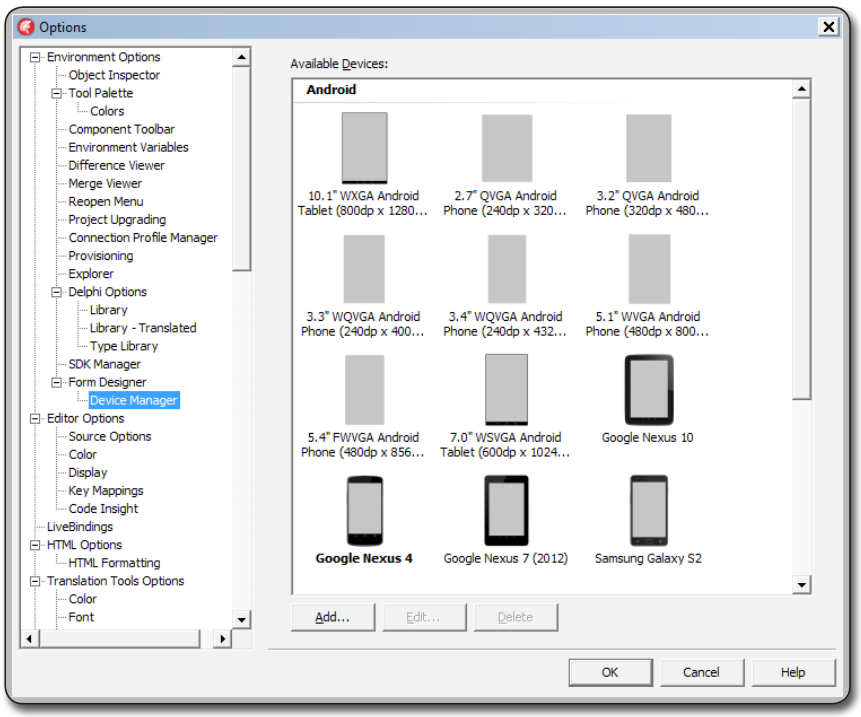

Figura 6. Device Manager

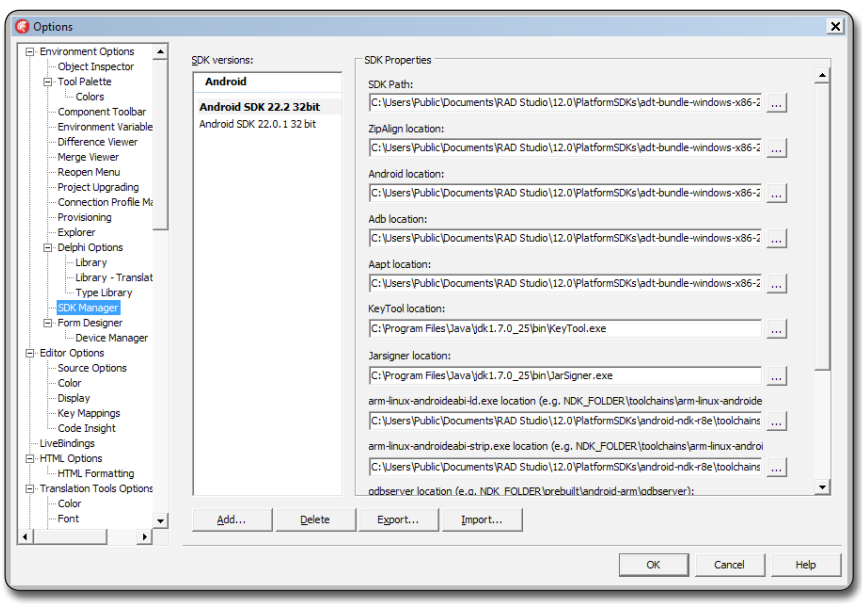

Figura 7. SDK Manager

SDK Manager

Ainda na seção Environment Options das opções da ferramenta, um novo item é adicionado, direcionado exclusivamente ao atual suporte a Android, denominado SDK Manager. Sua inclusão é justificada pelo fato do desenvolvimento para a plataforma Android envolver diferentes versões do sistema operacional e, consequentemente, diferentes versões de SDK. Por esta razão, agora o IDE do Delphi apresenta um gerenciador de SDKs, conforme visto na **Figura 7**.

MacinCloud Wizard

Com o efetivo acréscimo do suporte a iOS ainda na versão XE4, o Delphi se estabeleceu definitivamente como uma ferramenta de desenvolvimento de aplicações para as plataformas da Apple, vide o próprio iOS (Mobile) como também Mac OS X (PC). Conforme já mencionado, para este segmento torna-se necessária a utilização de um ambiente nativo Mac, em paralelo ao de desenvolvimento, para as questões de teste, execução e deploy das aplicações. Obviamente este tipo de requisito acaba por incorporar uma nova complexidade ao cenário de trabalho de determinados desenvolvedores, fundamentalmente aqueles que não possuem qualquer conhecimento sobre a plataforma exigida (Mac) ou não desfruta de meios para a obtenção da mesma. A fim de contribuir para este cenário, ainda no XE4, a Embarcadero disponibilizou a seus usuários um serviço relacionado, denominado MacinCloud, fruto de uma parceria comercial. Em suma, o MacinCloud se caracteriza por ser um serviço de aluguel de hardwares remotos, permitindo que seus utilizadores acessem, através da internet, determinados

servidores Mac. No contexto do Delphi, os desenvolvedores podem então fazer uso deste serviço como forma de construir, depurar, executar e fazer o deploy de suas aplicações para a plataforma Mac, em substituição ao ambiente local paralelo exigido.

Até então, tal serviço era disponibilizado sob a forma de um executável, responsável por configurar o IDE a se conectar ao serviço remoto. Agora no XE5 o MacinCloud se apresenta de forma nativa, tornando-se parte integrante do próprio IDE. O fruto disso é o surgimento de um novo assistente, denominado MacinCloud Wizard, disponibilizado no menu Help da ferramenta. Uma vez acionado, de imediato o assistente solicita as credenciais da EDN (Embarcadero Developer Network) do desenvolvedor (**Figura 8**), a fim compartilhá-la com o serviço do MacinCloud, tornando-o disponível para uso.

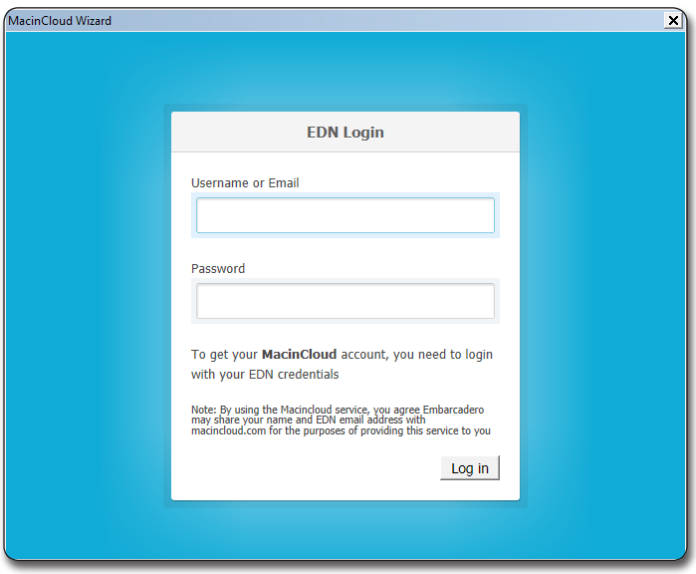

Figura 8. MacinCloud Wizard – Autenticação

Search Box

Ainda na versão 2010, o IDE do Delphi trouxe como uma de suas novidades, o IDE Insight que um poderoso recurso que possibilitava o desenvolvedor pesquisar por qualquer elemento presente no IDE, abrangendo de opções de configuração a componentes da Tool Palette. A inclusão de tal recurso pôde ser justificada pela natural evolução do ambiente de desenvolvimento, o que acabou por resultar num constante acréscimo de novas opções e funcionalidades ao longo de sua evolução. Sendo assim, o principal objetivo do IDE Insight era o de facilitar a utilização da própria ferramenta, por parte do desenvolvedor, evitando um eventual desperdício de seu tempo, a fim de se encontrar determinada opção ou elemento na ferramenta. Até o XE4, tal recurso era então provido por meio de uma janela pop-up, contendo uma caixa de pesquisa, acionada pelo menu Search>IDE Insight. Agora no XE5 o acionamento da funcionalidade do IDE Insight é "substituído" por uma simples caixa de pesquisa localizada no canto superior direito da ferramenta (**Figura 9**), denominada Search Box. Dessa forma, este recurso é agora parte integrante do layout padrão do ambiente, o que acaba por facilitar ainda mais seu acionamento, que se dá tanto pelo uso do menu tradicional (Search>IDE Insight), quanto por meio de teclas de atalho (F6 ou Ctrl + ponto (.)).

Figura 9. Search Box

Mudanças estruturais – System.Types

Esta nova versão do Delphi conta ainda com outra gama de novidades que podem ser consideradas mais "sutis", relacionadas às mudanças estabelecidas em alguns tipos (Types) de sua estrutura interna, pertencentes ao escopo System.Types. Um exemplo disso se dá sobre o type TVector, que nada mais é do que um record representativo de um vetor bidimensional, e que nesta versão do produto recebeu a adição de alguns novos métodos, tais como Rotate, MidVector e AngleCosine. O código da **Listagem 1** a seguir mostra um trecho de sua declaração.

Listagem 1. Declaração de TVector

```
TVector = record
  class function Create(const AX, AY, AW: Single):
   TVector; overload; static; inline;
  class function Create(const APoint: TPointF):
   TVector; overload; static; inline;
...
  function Rotate(const AAngle: Single): TVector; inline;
  function MidVector(const AVector: TVector): TVector; inline;
  function AngleCosine(const AVector: TVector): Single;
...
end;
```

De forma similar, outro tipo afetado é o TMatrix, que é um record representativo de uma matriz de vetores, usado em transformações vetoriais em 2D. Neste caso, sua estrutura ganha o acréscimo de três novos construtores (CreateRotation, CreateScaling e CreateTranslation) e de alguns métodos de classe, tais quais os nomeados como Inverse e Determinant. Em complemento, enquanto as alterações em determinados tipos refletem somente no acréscimo de itens à sua estrutura, para outros tais mudanças incluem também a retirada/troca de alguns de seus elementos. Um exemplo é o tipo TMatrix3D, que representa uma matriz 4x4 a ser utilizada em contextos 3D. Agora no XE5, de sua definição foram retirados os métodos TransposeMatrix3D e ScaleMatrix3D, os quais foram substituídos pelos equivalentes Transpose e Scale, conforme mostrado na **Listagem 2**.

Por fim, ainda seguindo por esta linha, o tipo TVector3D que atua como uma espécie de classe para vetores em 3D, teve alguns de seus métodos anteriormente existentes (AddScale e AddVector3D) retirados de sua definição atual. Obviamente, a lista de mudança inclui ainda outros tipos, cuja totalidade por ser vista no próprio Help online do produto.

Listagem 2. Declaração de TVector3D

```
TVector3D = record
private
  function DetInternal(const a1, a2, a3, b1,
   b2, b3, c1, c2, c3: Single): Single; inline;
  function Scale(const AFactor: Single): TMatrix3D;
public
  function Transpose: TMatrix3D;
  function Determinant: Single;
...
end;
```

FireDAC

O FireDAC é uma biblioteca de componentes de acesso a dados que provê acesso direto, nativo e de alta performance a diversas fontes de dados, incluindo os principais SGBDs do mercado – Firebird, InterBase, MySQL, DB2, Oracle, SQL Server, PostgreSQL, entre outros. Originário de um produto de terceiro (AnyDAC, da DA-Soft Technologies) adquirido pela Embarcadero, o FireDAC surgiu como uma novidade ainda na versão XE3 da ferramenta. Até então, ele era disponibilizado ao desenvolvedor por meio de um instalador à parte. Agora na versão XE5, a biblioteca é efetivamente integrada ao contexto da ferramenta, fazendo parte da própria instalação do IDE. Isso resulta em novas guias nativas na Tool Palette, conforme mostra a **Figura 10**.

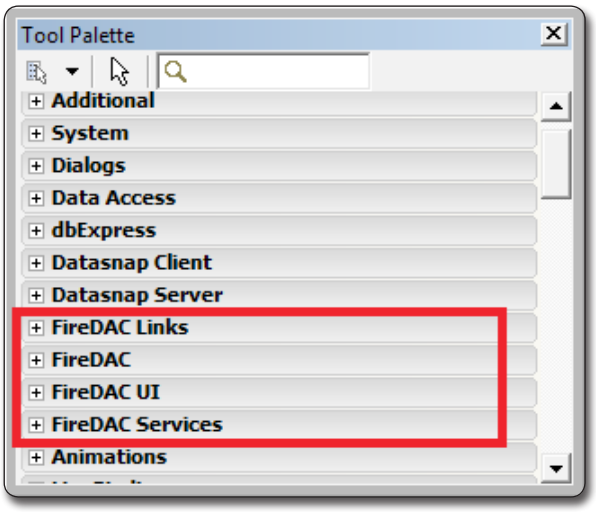

Figura 10. Tool Palette – FireDAC

CURSOS ONLINE

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

Clube Delphi

CONHEÇA ALGUNS DOS CURSOS:

- Curso de Multicamadas com Delphi e DataSnap
- Delphi para Iniciantes
- Criando componente Boleto em Delphi
- Loja Virtual em Delphi Prism

Para mais informações :
www.devmedia.com.br/cursos/delphi
(21) 3382-5038

DEVMEDIA

Além disso, a biblioteca FireDAC está para as aplicações móveis (FireMonkey Mobile Application) assim como o dbExpress estava para as aplicações Desktop (VCL Forms Application) em tempos recentes, tornando-se a principal solução de acesso a dados para o contexto.

iOS 7

Conforme já mencionado, a versão XE5 do Delphi conta ainda com algumas novidades não relacionadas diretamente a Android. Uma delas é o suporte atualizado a iOS, mais especificamente à sua versão 7 (iOS 7), a nova versão do sistema operacional de dispositivos móveis da Apple (vide iPhone, iPod Touch e iPad). No IDE isso se traduz em novos estilos FireMonkey (FireMonkey iOS Style) e arquivos de recursos condizentes. Em vista desse suporte, a construção de aplicações móveis para iOS 7 acaba por envolver novas nuances. Uma delas está relacionada especificamente à construção de soluções direcionadas à loja de aplicativos oficial da fabricante, a Apple App Store. Neste cenário, torna-se requisito essencial à adição de dois novos ícones de aplicação, no formato PNG, tidos respectivamente como "padrão" e "de alta resolução", conforme mostrado a seguir:

- Resolução do ícone "padrão" para dispositivos iPhone e iPod: 60x60;
- Resolução do ícone "padrão" para dispositivos iPad: 76x76;
- Resolução do ícone "de alta resolução" para dispositivos iPhone e iPod: 120x120;
- Resolução do ícone "de alta resolução" para dispositivos iPad: 152x152.

Tecnologias Web

Ao longo da extensa trajetória do Delphi algumas tecnologias voltadas para a Web foram sendo incorporadas à ferramenta. WebSnap, Internet Express, IntraWeb (VCL for the Web) e WebBroker (ver **BOX 2**) podem então ser consideradas como quatro das principais, estando envolvidas em diferentes versões do IDE. Até o XE4, todas elas ainda se mostravam presentes, seja pela aceitação e usabilidade adquiridas ao longo dos anos, ou apenas por motivos de retro compatibilidade com projetos existentes. De qualquer forma, no XE5 este cenário muda de forma definitiva, com a remoção de duas destas tecnologias do contexto do IDE: WebSnap e Internet Express. Isso significa dizer que a partir de agora, a própria instalação do produto já não contempla ambas, mantendo apenas as tecnologias IntraWeb e WebBroker para o desenvolvimento Web atual com Delphi.

BOX 2. WebBroker

No Delphi, os componentes WebBroker ficam localizados no grupo Internet da paleta de componentes do IDE. De forma básica, sua forma de atuação e todo seu processamento envolvem a criação de manipuladores de eventos que estarão associados a uma URI (Uniform Resource Identifier). Adicionalmente, seu uso também se relaciona à definição de documentos compatíveis com a Web (Ex: HTML e XML) a serem transferidos à parte cliente.

Conclusão

Este artigo apresentou uma visão geral das principais novidades da mais nova versão do Delphi, a XE5, cujo foco se dá sobre o desenvolvimento móvel para a plataforma Android. Este cenário se assemelha ao apresentado em seu release anterior, no lançamento da versão XE4, onde grande parte das novidades se voltavam ao então recém promovido suporte nativo ao desenvolvimento móvel para iOS. Ainda de posse de tal comparativo, da mesma maneira que o produto foi anteriormente batizado de forma comercial como "Delphi for iOS", esta nova versão é tratada como "Delphi for Android".

Isto posto, no estágio atual da ferramenta, é possível afirmar que ela consolida seus recentes adjetivos de multiplataforma (Windows, Mac OS X, iOS e Android) e multidispositivo (PCs, Tablets e Smartphones), uma vez que possibilita o desenvolvimento de aplicações para contextos distintos, a partir de uma mesma base de código e de conhecimento. Assim como ocorre a cada nova versão, uma gama de novos desafios está lançada! Cabe a você, desenvolvedor, saber tirar proveito das novas possibilidades que a ferramenta lhe oferece.

Autor

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ e pós-graduado em Engenharia de Componentes. Atua como Analista Programador Delphi há 8 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas. Certified Delphi Developer. Certified Embarcadero Instructor. Site pessoal: www.fabriciokawata.com.br

Links:

Delphi XE5 – Download Trial
https://www.embarcadero.com/br/products/delphi/downloads

Delphi XE5 – Página oficial do produto
https://www.embarcadero.com/br/products/delphi

MacinCloud - Página oficial
http://www.macincloud.com

Android - Página relacionada ao desenvolvimento para a plataforma
http://developer.android.com

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Introdução ao UniDAC – Parte 2

Manipulando dados com os componentes da biblioteca UniDAC

Este artigo faz parte de um curso

O UniDAC, como produto, se apresenta como uma extensa biblioteca de componentes de acesso a dados e que dispõe de capacidade para trabalhar com diversas fontes de dados diferentes, essencialmente bancos de dados e fontes ODBC. Dentre os SGBDs suportados estão inclusas as principais opções de mercado, tais como Oracle, Firebird, InterBase, Microsoft SQL Server, PostgreSQL e MySQL. Por este fato, o UniDAC torna-se uma opção real de uso aos desenvolvedores, atendendo a diversos segmentos comerciais. Em vista de justificar a sua adoção, no primeiro artigo foram explicitadas suas principais características, que naturalmente o colocam no mesmo nível das opções nativas dbExpress e FireDAC (ver **BOX 1**). Adicionalmente, foram abordados também alguns de seus recursos peculiares, que estabelecem seu diferencial mediante seus concorrentes.

BOX 1. dbExpress e FireDAC

dbExpress e FireDAC são atualmente as duas principais tecnologias nativas presentes no cenário Delphi para a parte de acesso a banco de dados. No popular, são tidas como frameworks ou bibliotecas, uma vez que são compostas por um conjunto de classes, drivers e componentes direcionados à conectividade a dados. A primeira (dbExpress), introduzida ainda na época da versão 6 do IDE, já é bastante conhecida da comunidade e tem como atrativos, sua facilidade de uso e a vasta disponibilidade de materiais e informações na grande rede. Já o FireDAC foi inserido no contexto Delphi na versão XE3 do IDE, após a aquisição do produto (AnyDAC, da DA-Soft) por parte da Embarcadero. Seu grande atrativo se reflete em sua conhecida excelência nas tratativas de acesso a dados, em pontos como performance e suporte a recursos específicos a diversos SGBDs. Em meio a comunidade, alguns se arriscam a dizer que "o FireDAC é tudo que o dbExpress sonhou em ser um dia".

Fique por dentro

São expostos os aspectos práticos da utilização do UniDAC na construção de aplicações de banco de dados. Se na primeira parte do artigo o intuito foi fazer um overview do produto como um todo, nesta segunda (e última) parte, o objetivo se dá pela exposição do uso de seus componentes a partir de exemplos construídos no Delphi. No contexto final, tal abordagem poderá servir como um guia inicial para o desenvolvedor que optar pela utilização da biblioteca em seus projetos. Também pode ser de grande ajuda para aqueles que já possuem aplicações desenvolvidas com outros componentes, como é o caso do dbExpress, ADO, IBX ou soluções de terceiros, tornando o processo de substituição e uso destes novos componentes mais natural e menos traumático.

No presente artigo, o foco se dá sobre o lado prático de uso da biblioteca, mais especificamente de seus componentes, em aplicações de banco de dados no Delphi.

Considerações iniciais

Por envolver um cenário com uma abordagem mais prática, torna-se necessário o estabelecimento de alguns pontos iniciais. O primeiro deles diz respeito à versão do IDE a ser utilizada. Para o trabalho atual, a versão escolhida é a XE4. Já olhando especificamente para a parte de banco de dados, o SGBD abordado será o Firebird (versão 2.5), por se tratar de uma opção gratuita e em razão de sua popularidade em meio à comunidade Delphi. Por fim, em se tratando do UniDAC, o artigo se baseia em sua versão 5.0, a mais recente até o momento da escrita deste texto. No site oficial do produto (ver seção **Links**), esta é nominalmente apresentada como "UniDAC 5.0 for RAD Studio XE4".

O processo de instalação do UniDAC é tão natural quanto de qualquer outro aplicativo Windows, provido por um Wizard que envolve os tradicionais passos "Next-Next-Finish". Dessa forma, neste primeiro momento, não se faz necessária nenhuma observação adicional. Apenas como forma de ilustrar a instalação

em si, a **Figura 1** mostra a etapa inicial do processo. Na imagem, é possível ver então os detalhes da versão (5.0.2) e edição (Trial Edition) utilizadas do produto.

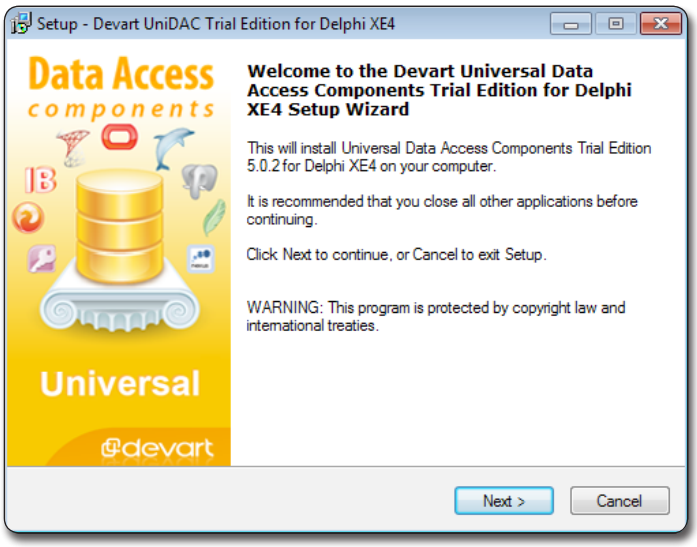

Figura 1. Tela inicial de instalação do UniDAC

Conforme já mencionado, o UniDAC é um produto comercial (pago), no entanto, assim como a maioria dos softwares comerciais, ele provê uma versão gratuita experimental – as já famosas edições Trial – que permitem um acesso irrestrito ao produto durante determinado período, comumente de 30 dias. No caso do UniDAC, esse período é um pouco maior (60 dias), conforme mostra a **Figura 2**. Tal mensagem é exibida no momento da primeira execução do IDE, uma vez que a biblioteca tenho sido instalada.

Figura 2. UniDAC Trial Edition – 60 dias

Ponto de partida – conectando ao banco de dados

Naturalmente, o ponto de partida para o uso efetivo do UniDAC na construção de aplicações de banco de dados no Delphi está no estabelecimento da conexão com o banco envolvido. Para este artigo, a fim de se manter o foco sobre a biblioteca, será utilizado como fonte de dados a já conhecida base de exemplo que acompanha a instalação do Firebird, denominada "employee.fdb". De posse do banco de dados, o componente UniDAC responsável pela conexão é o TUniConnection. Assim, tendo como base uma aplicação VCL Forms tradicional, tal componente envolve algumas configurações, mais especificamente o conjunto de parâmetros de conexão. Independente do SGBD envolvido boa parte destes parâmetros são comuns a todos, conforme listado a seguir:

- Provider: especifica o provedor (provider) de dados a ser utilizado pela conexão. No caso do Firebird, utiliza-se a opção InterBase;
- Username: nome do usuário que será utilizado no processo de autenticação no servidor de banco de dados;
- Password: senha relacionada ao usuário e que será utilizada no processo de autenticação no servidor de banco de dados;
- Server: especifica o nome ou o endereço IP da máquina onde está localizado o servidor de banco de dados. Para bancos Firebird locais, comumente utiliza-se o valor "localhost";
- Database: no caso de InterBase e Firebird, especifica o caminho físico do banco de dados;
- Port: determina o número da porta a ser utilizada pela conexão.

Além destas configurações tidas como "padrão", ao se trabalhar com um banco de dados Firebird é necessário realizar mais dois ajustes iniciais essenciais. O primeiro deles diz respeito à definição do tipo de protocolo que será utilizado na conexão, uma vez que este SGBD provê três opções: TCP/IP, NetBEUI e SPX. Adicionalmente, deve-se especificar também a biblioteca cliente de banco de dados (Client Library) que estará relacionada à conexão. Isto se deve ao fato do provider de dados "InterBase" servir tanto para bancos InterBase quanto para Firebird. Sendo assim, torna-se necessário a especificação da biblioteca gds32.dll para o primeiro, ou fbclient.dll para o segundo. Tais parâmetros são disponibilizados por meio da propriedade SpecificOptions do componente que, ao ser acionada, exibe uma caixa de diálogo, conforme mostrado na **Figura 3**.

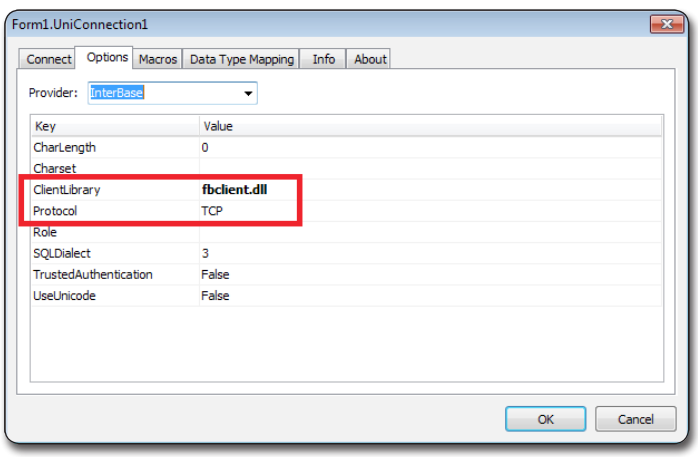

Figura 3. TUniConnection – Propriedade EspecificOptions

Por fim, a **Listagem 1** mostra o código dfm do componente já configurado para o exemplo. A propriedade Connected setada como True denota a efetivação da conexão.

Recuperando dados de uma tabela do banco de dados

Durante o desenvolvimento de uma aplicação de dados no Delphi, uma vez de posse da conexão com o banco de dados,

o próximo passo natural se faz pela recuperação dos dados de determinada tabela deste. Com o UniDAC isso se dá pela utilização do componente TUniQuery que, conforme seu nome sugere, possui a mesma intenção dos tradicionais componentes "Query" nativos, tais como TIBQuery (guia InterBase) e TSQLQuery (guia dbExpress).

Para fins demonstrativos, a intenção aqui se resume a recuperar os dados da tabela EMPLOYEE do banco utilizado como exemplo. Pelo fato de sua estrutura envolver uma série de campos, somente alguns deles serão selecionados. Em termos de configuração do componente, para este cenário, apenas duas de suas propriedades estão envolvidas: Connection e SQL. A primeira recebe uma conexão ativa, representada por um elemento do tipo TUniConnection. Já a segunda, do tipo TStrings, receberá a instrução SQL a ser executada no servidor de banco de dados, para a recuperação das informações provenientes da tabela. Dito isto, a **Listagem 2** expõe um código de amostra que efetivamente cria um objeto TUniQuery em runtime, já voltado ao contexto do exemplo.

Listagem 1. Configurações do componente TUniConnection

```
01 object UniConnection1: TUniConnection
02   ProviderName = 'InterBase'
03   Port = 3050
04   Database = '<diretório>\EMPLOYEE.FDB'
05   SpecificOptions.Strings = ('InterBase.ClientLibrary=fbclient.dll')
06   Username = 'SYSDBA'
07   Password = 'masterkey'
08   Server = 'localhost'
09   ...
10 end
```

Listagem 2. Criação de um objeto TUniQuery em runtime

```
01 var
02   MyQuery: TUniQuery;
03 begin
04   MyQuery := TUniQuery.Create(nil);
05   MyQuery.Connection := UniConnection1;
06   MyQuery.SQL.Add('select EMP_NO, FIRST_NAME, LAST_NAME, JOB_COUNTRY,
       SALARY');
07   MyQuery.SQL.Add('from EMPLOYEE order by EMP_NO');
08   MyQuery.Open;
09 end;
```

No aspecto técnico, partindo do princípio da existência de um TDBGrid (DBGrid1) no formulário, somente o código apresentado não é o suficiente para se expor os dados em tela. Como complemento, essencialmente é necessária a inclusão das instruções responsáveis pela efetivação do "binding" entre o objeto TUniQuery e o controle Grid, provido por um elemento TDataSource. Em vista disso, a **Listagem 3** exibe o trecho relacionado que complementa o código anterior.

Na linha 4 é então criado o objeto do tipo TDataSource (MyDataSource), que será utilizado para fazer a ligação entre o objeto TUniQuery (MyQuery) e o TDBGrid (DBGrid1), conforme mostrado nas linhas 5 e 6 respectivamente. Por fim, a **Figura 4** mostra visualmente o resultado final de todo este processo.

Listagem 3. Criação de um TDataSource em runtime

```
01 var
02   MyDataSource: TDataSource;
03 begin
04   MyDataSource := TDataSource.Create(nil);
05   MyDataSource.DataSet := MyQuery;
06   DBGrid1.DataSource := MyDataSource;
07 end;
```

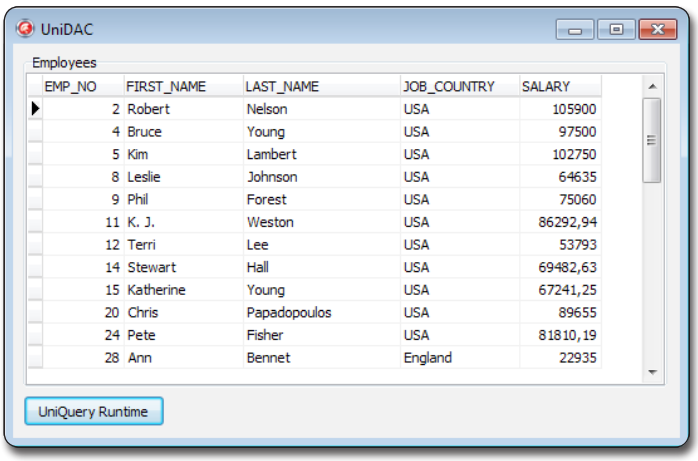

Figura 4. TUniQuery – Dados da tabela Employee sendo mostrados em tela

Provider UniDAC

Uma vez realizada todas as configurações dos componentes de acesso a dados e de tela (controles), em teoria, a aplicação está apta a rodar. No caso do exemplo abordado, o resultado esperado é refletido na figura mostrada anteriormente (**Figura 4**), todavia, na imediata tentativa de execução da mesma, uma exceção (exception) é levantada, tal qual mostrada na **Figura 5**.

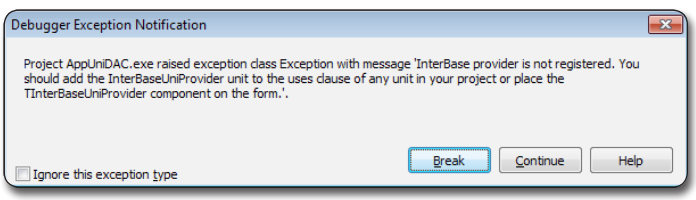

Figura 5. Exception lançada no momento da primeira execução

Apenas para lembrar, a estrutura provida pelo UniDAC para o trabalho com banco de dados necessita de um elemento provider específico ao SGBD com o qual se está trabalhando. Em função disso, uma vez que a biblioteca é instalada, são disponibilizados elementos provider específicos para cada fonte suportada, estando disponíveis no grupo UniDAC Providers, da Tool Palette do IDE. Tendo isso em mente, os dizeres apresentados pela notificação de exceção, em suma, indicam a ausência de um provider adequado registrado na aplicação. Justamente pelo fato do SGBD utilizado ser o Firebird e o provider selecionado no componente TUniConnection ser o do InterBase, a mensagem faz menção a este ponto: "InterBase provider is not registered". Além de apontar o

problema, a própria mensagem provê ainda alternativas para a solução do mesmo. A mais natural se faz pelo acréscimo de um componente TInterBaseUniProvider ao Form ou, simplesmente pela adição da unit InterBaseUniProvider na seção uses do código. Tomadas às devidas providências, a aplicação finalmente se torna apta a ser executada tal qual qualquer outra aplicação de banco de dados construída no Delphi.

Relacionamento mestre-detalhe (Master-Detail)

Ainda relacionado à recuperação de dados de tabelas, outro aspecto muito comum no desenvolvimento de aplicações de banco de dados se dá pelo trabalho com tabelas relacionadas, tal qual o caracterizado por um relacionamento Master-Detail (mestre-detalhe) entre as mesmas. Com o UniDAC, este tipo de abordagem é naturalmente muito semelhante aos de outras bibliotecas, onde o elemento "Mestre" se liga ao elemento "Detalhe" por meio de um objeto intermediário do tipo TDataSource. No intuito de exemplificar esta ação, o próprio banco de dados adotado como exemplo provê tal tipo de relacionamento entre tabelas. Para este cenário, serão então utilizadas as tabelas EMPLOYEE e EMPLOYEE_PROJECT, onde a primeira fará o papel de "Mestre", enquanto a segunda tabela fará o de "Detalhe". Em termos de estrutura, ambas se relacionam por meio do campo EMP_NO, conforme pode ser visto seguir, onde são exibidos trechos do código DDL de cada objeto do banco. Pelo lado funcional, EMPLOYEE nada mais é do que a tabela que guarda as informações dos empregados da empresa, enquanto que EMPLOYEE_PROJECT detêm os eventuais projetos relacionados a cada um deles (**Listagem 4**).

Listagem 4. Descrição das tabelas envolvidas

```
/* Tabela Employee (Master) */
CREATE TABLE EMPLOYEE (
  EMP_NO      EMPNO NOT NULL,
  FIRST_NAME   FIRSTNAME NOT NULL,
  LAST_NAME    LASTNAME NOT NULL,
    ...
);

/* Tabela Employee Project (Detail) */
CREATE TABLE EMPLOYEE_PROJECT (
  EMP_NO   EMPNO NOT NULL,
  PROJ_ID  PROJNO NOT NULL);

ALTER TABLE EMPLOYEE_PROJECT
  ADD FOREIGN KEY (EMP_NO) REFERENCES EMPLOYEE
 (EMP_NO);
```

Voltando para o lado prático, para se reproduzir o referido relacionamento na aplicação são utilizados dois componentes TUniQuery, sendo um para cada tabela. Logo, para o elemento "Mestre", exatamente o mesmo processo efetuado no tópico anterior pode ser utilizado. Contudo, em complemento à abordagem em runtime apresentada anteriormente, aqui todo o processo de configuração se dará em design-time. Dito isto, a seguir é mostrado o código do objeto TUniQuery para EMPLOYEE, nomeado como QueryMaster (**Listagem 5**).

Listagem 5. Visualização do dfm do componente QueryMaster

```
object QueryMaster: TUniQuery
  Connection = UniConnection1
  SQL.Strings = (
    'select EMP_NO, FIRST_NAME, LAST_NAME, JOB_COUNTRY, SALARY '
    'from EMPLOYEE' 'order by EMP_NO')
...
end
```

Conforme já mencionado, TDataSource é o tipo do objeto responsável por estabelecer a ligação entre mestre e detalhe. Neste exemplo, tal objeto, nomeado aqui como DataSourceMaster, se liga ao componente QueryMaster através de sua propriedade DataSet, conforme o código a seguir:

```
object DataSourceMaster: TDataSource
  DataSet = QueryMaster
...
end
```

Por fim, a configuração elemento TUniQuery para a parte "Detail" é mostrada no código da **Listagem 6**.

Listagem 6. Visualização do dfm do componente QueryDetail

```
object QueryDetail: TUniQuery
  Connection = UniConnection1
  SQL.Strings = (
    'select E.EMP_NO, E.PROJ_ID, P.PROJ_NAME, P.PROJ_DESC'
    'from EMPLOYEE_PROJECT E'
    'inner join PROJECT P on(P.PROJ_ID = E.PROJ_ID)'
    'order by E.PROJ_ID')
  MasterSource = DataSourceMaster
  MasterFields = 'EMP_NO'
  DetailFields = 'EMP_NO'
...
end
```

O principal comentário se faz aqui sobre a propriedade MasterSource que, conforme o nome indica, recebe o elemento DataSource que está vinculado ao DataSet "Mestre". Em complemento, MasterFields e DetailFields indicam o nome dos Fields de referência em cada estrutura que, para este caso, conforme já mencionado, se trata do campo EMP_NO. Através destas indicações é então estabelecida a devida relação de "mestre e detalhe" dos componentes TUniQuery envolvidos. Ainda sobre a configuração apresentada, de forma complementar, foi atribuída à propriedade SQL uma instrução envolvendo JOIN, a fim de se recuperar detalhes do Projeto ao qual o empregado esteja relacionado. Isto porque, a tabela EMPLOYEE_PROJECT apenas armazena a referência do projeto em questão, que aponta para a tabela PROJECT, onde são mantidos os detalhes do mesmo. Sendo assim, através da junção aplicada à instrução SQL é possível se recuperar as informações como Nome (PROJ_NAME) e Descrição (PROJ_DESC) do projeto. Por fim, a **Figura 6** mostra a aplicação de exemplo já contemplando toda a parte mestre-detalhe.

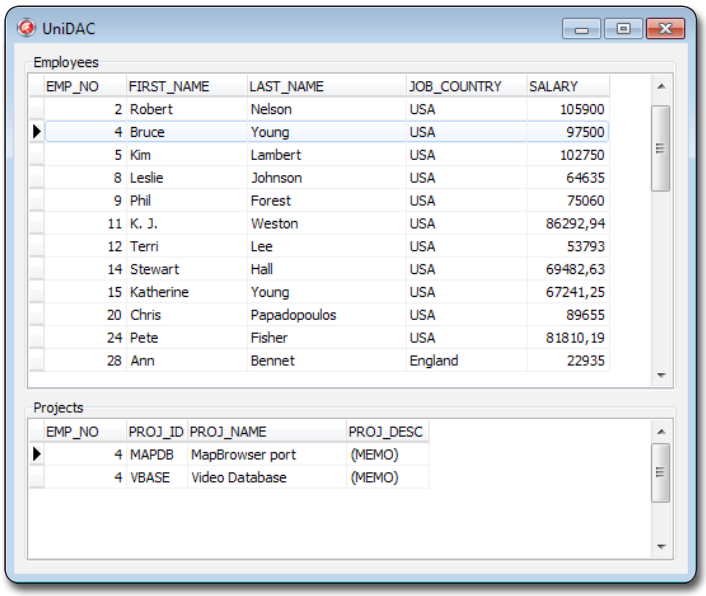

Figura 6. Relacionamento mestre-detalhe com UniDAC

Atualizando dados

Com o UniDAC, o processo de atualização de dados de uma determinada tabela do banco de dados, de uma forma geral, é algo tão trivial quanto o realizado com outras bibliotecas de acesso a dados. Tomando como exemplo novamente a utilização de seu componente TUniQuery, a atualização direta dos dados da tabela Employee, como é mostrado na **Figura 4**, se dá pelo simples editar das informações dispostas nas colunas do grid. De forma simples, uma vez que determinada coluna é modificada, ao sair dela, o método Post é automaticamente chamado, fazendo com que toda e qualquer alteração seja imediatamente salva no banco. Apesar de funcional, o processo descrito é considerado, mediante o contexto do UniDAC, uma forma ineficiente de atualização de dados. Isto porque, toda atualização de dados envolve a produção, seja de forma implícita ou explícita, de uma instrução SQL de Update. Esta, por sua vez, se torna otimizada ao ter sua cláusula Where sendo formada pelo(s) campo(s) chave da tabela (primary key), o que contribui para a localização do registro a ser atualizado pelo próprio mecanismo de banco de dados. A seguir na **Listagem 7** é mostrado um exemplo de instrução otimizada de Update para a tabela Employee.

Listagem 7. Instrução Update

```
01 UPDATE EMPLOYEE SET
02   FIRST_NAME = 'Robert',
03   LAST_NAME = 'Nelson',
04   SALARY = 105900
05 WHERE (EMP_NO = 2);
```

Dito isto, em se tratando especificamente de um banco Firebird, com base no processo citado, internamente o componente TUniQuery tenta recuperar as informações sobre chave primária da tabela executando uma consulta adicional no banco de dados. Isso se reflete em uma operação a mais a cada ação de Update o que, inevitavelmente, resulta numa natural perda de performance. Em vista disso, o próprio componente provê meios para se evitar tal situação. A mais imediata envolve o uso de sua propriedade KeyFields, de tipo string, que irá receber os nomes dos campos da tabela a serem usados como chave e que serão utilizados na montagem das eventuais instruções de Update. Sendo assim, o código apresentado na **Listagem 2**, para a criação do componente TUniQuery, poderia ser reescrito de forma ideal conforme mostrado na **Listagem 8**.

Listagem 8. Código ideal para a criação de um objeto TUniQuery em runtime

```
01 var
02   MyQuery: TUniQuery;
03 begin
04   MyQuery := TUniQuery.Create(nil);
05   MyQuery.Connection := UniConnection1;
06   MyQuery.KeyFields := 'EMP_NO';
07   MyQuery.SQL.Add('select EMP_NO, FIRST_NAME, LAST_NAME, JOB_COUNTRY,
     SALARY');
08   MyQuery.SQL.Add('from EMPLOYEE order by EMP_NO');
09   MyQuery.Open;
10 end;
```

Controle sobre as instruções SQL geradas

A utilização de um componente do tipo TUniQuery durante o desenvolvimento de uma aplicação de banco de dados no Delphi está relacionada, de uma forma geral, a todas as operações sobre dados tradicionalmente conhecidas. Nominalmente, tais operações são enumeradas como Select, Delete, Insert, Lock, Refresh e Update. Em vista disso, o componente TUniQuery provê, através de suas propriedades, uma forma de controle exato sobre as instruções SQL relacionadas a tais operações. Sendo assim, por meio de suas propriedades SQL, SQLDelete, SQLInsert, SQLLock, SQLRefresh e SQLUpdate o desenvolvedor tem um controle exato sobre cada instrução SQL a ser utilizada pelo componente. Ao contrário do exemplo anteriormente mostrado, onde o comando Update utilizado pelo componente foi gerado de forma automática e dinâmica pelo mesmo, aqui os comandos envolvidos são providos de forma estática pelo desenvolvedor.

Na prática, este processo se dá ainda em tempo de design, mediante o preenchimento das propriedades citadas. Obviamente, num desenvolvimento real, o preenchimento manual de cada uma delas se torna um tanto quanto trabalhoso e pouco produtivo. A fim de contornar esta situação, o próprio componente é capaz de autogerar suas instruções, preenchendo de forma automática cada uma dessas propriedades. Esta facilidade se dá por meio de um editor, acessível por meio de um duplo-clique sobre o próprio componente. Dentre as diversas abas contidas nesta estrutura, duas delas são as responsáveis por todo o processo. A primeira, denominada SQL, é onde deverá ser informada a instrução SQL principal, que envolve a recuperação dos dados da tabela desejada do banco de dados, tal qual já realizado nos exemplos anteriores. A segunda, denominada SQL Generator, é onde serão efetiva-

mente geradas as demais instruções relacionadas, por meio do acionamento do botão Generate SQL, conforme mostrado na **Figura 7**. Nesta exemplificação, o componente TUniQuery está direcionado à tabela Country do banco de dados.

Para confirmar, na **Listagem 9** é exibido um trecho de código do componente onde podem ser vistas algumas de suas propriedades relacionadas a SQL, já devidamente preenchidas imediatamente após a configuração realizada em seu editor.

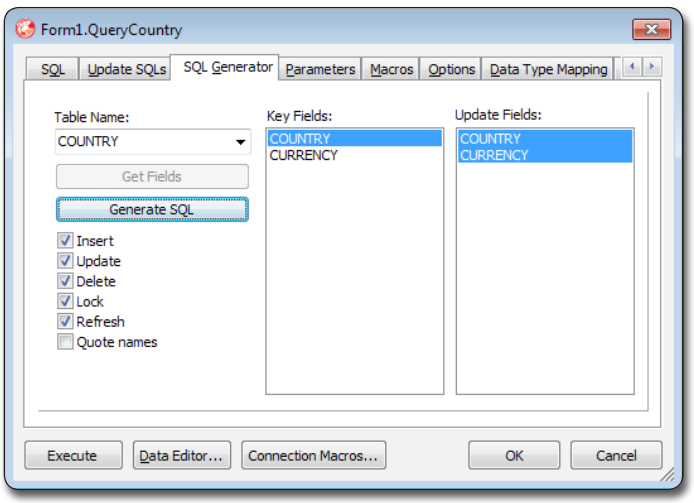

Figura 7. Editor SQL do componente TUniQuery

Listagem 9. Preenchimento das propriedades relacionadas à SQL no componente QueryCountry

```
01 object QueryCountry: TUniQuery
02   SQLInsert.Strings = (INSERT INTO COUNTRY
     (COUNTRY, CURRENCY) VALUES
      (:COUNTRY, :CURRENCY)')
03   SQLDelete.Strings =
     ('DELETE FROM COUNTRY WHERE COUNTRY =
      :Old_COUNTRY')
04   SQLUpdate.Strings = ('UPDATE COUNTRY
       SET COUNTRY = :COUNTRY, CURRENCY =
       :CURRENCY WHERE COUNTRY = :Old_COUNTRY')
05 ...
06 end
```

Operando em modo desconectado

No Delphi, o uso do dbExpress em aplicações de banco de dados proporciona uma gama de recursos diversos, relacionados tanto ao tratamento e manipulação dos dados, quanto da própria conexão. Para esta última, um dos recursos providos é o modo de operação "desconectado" com o banco de dados, útil em casos onde há certa debilitação do servidor de dados ou frequente instabilidade na rede em que o aplicativo opera. Tecnicamente, este recurso produz um cenário onde uma conexão efetiva com o banco só é aberta quando necessária, essencialmente em operações que realizem algum tipo de chamada ao servidor (Ex: instruções de Select, Insert, Update e Delete). Por conseguinte, uma vez que a operação é concluída, a conexão se fecha de forma automática. Na prática, o responsável por todo este controle fica a cargo do componente de conexão do dbExpress, o TSQLConnection, através de sua propriedade KeepConnection. Por padrão, ela vem como True, indicando um comportamento contrário ao modo desconectado, onde a conexão com o banco, uma vez ativada é permanentemente mantida. Isso ocorre fundamentalmente por três motivos:

- Por si só, o dbExpress considera um cenário otimizado de atuação para aplicação, onde dificuldades de rede e de servidor não são críticos;
- Pelo fato de uma aplicação de banco de dados tradicionalmente primar por operações CRUD (Create-Retrieve-Update-Delete) de dados, a manutenção permanente da conexão com o banco agiliza todo o processo;
- Operar no modo desconectado pode se tornar muito custoso em vários aspectos à aplicação, pelo fato de que cada operação de banco de dados irá exigir uma conexão/desconexão com o mesmo.

Dito isto, o UniDAC também provê de um recurso similar ao do dbExpress para operar em modo desconectado. Da mesma forma, o componente de conexão (TUniConnection) é quem centraliza tal configuração por meio de sua propriedade Options.DisconnectedMode (**Figura 8**). Sendo assim, uma vez marcada como True, toda e qualquer operação a ser realizada sobre o banco de dados envolverá a abertura de uma nova conexão e o subsequente fechamento da mesma. De forma complementar, caso a conexão tenha sido explicitamente aberta (por meio do método Connect ou atribuindo True à propriedade Connected) ela se mantém até que seja também explicitamente fechada (por meio do método Disconnect ou atribuindo False à propriedade Connected).

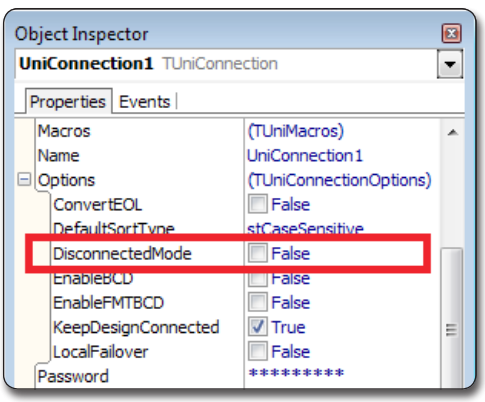

Figura 8. Propriedade DisconnectedMode

Utilizando transações

Outro ponto comum e importante neste tipo desenvolvimento é o uso de transações para controle efetivo das ações a serem realizadas pela aplicação com o banco de dados. A utilização destas se torna essencial em cenários onde uma sequência de operações deve ser tratada como um único bloco atômico, prevenindo eventuais falhas. No popular, ou todas as operações são realizadas com sucesso ou todas são abortadas (em caso de falha em alguma). No UniDAC, o controle transacional ocorre de forma local, sendo refletido automaticamente no servidor. Seu gerenciamento fica a

cargo do componente de conexão TUniConnection, por meio de alguns métodos direcionados a cada etapa do processo como um todo. Alguns destes são descritos a seguir:

- StartTransaction: inicia uma nova transação no servidor;
- Commit: confirma todas as alterações (insert, update e delete) de dados da transação atual de forma permanente no banco de dados, encerrando-a a seguir;
- Rollback: descarta todas as alterações (insert, update e delete) de dados da transação atual, encerrando-a a seguir;
- CommitRetaining: semelhante ao Commit, com o diferencial de que mantém a transação ativa ao final do processo;
- RollbackRetaining: semelhante ao Rollback, com o diferencial de que mantém a transação ativa ao final do processo.

Em se tratando especificamente dos comandos CommitRetaining e RollbackRetaining, tecnicamente eles refletem apenas uma ação resumida de subsequentes chamadas a StartTransaction e Commit/Rollback. Em outras palavras, a inexistência destes comandos poderia ser perfeitamente suprida pelos demais existentes. Tanto CommitRetaining quanto RollbackRetaining são nativamente suportados pelos SGBDs Firebird e InterBase, enquanto que para os demais servidores de dados, tais funcionalidades são apenas emuladas, em processos similares aos citados anteriormente.

Olhando pelo lado prático do uso do UniDAC, antes de se iniciar uma nova transação é de suma importância garantir que nenhuma outra esteja ativa, evitando assim o lançamento de uma exceção em runtime. Essa garantia é conseguida através de uma simples verificação de uma propriedade específica do componente, denominada InTransaction, que retorna se há (True) ou não (False) uma transação corrente, tal como mostrado a seguir:

```
if not UniConnection1.InTransaction then
...
```

DataSet Manager

Além dos componentes que são instalados no IDE, o UniDAC provê ainda algumas ferramentas adicionais que também se anexam ao ambiente de desenvolvimento do Delphi. Dentre estas, está o DataSet Manager que, conforme seu nome sugere, nada mais é do que um gerenciador de DataSets para projetos de aplicações de banco de dados. Por padrão, após a instalação da biblioteca, o DataSet Manager é disponibilizado na parte esquerda do IDE, ao lado na janela Structure (**Figura 9**). De forma prática, com ele, todo elemento DataSet, provindo do UniDAC ou não, presente no projeto corrente é automaticamente mapeado para sua estrutura. A partir disso, sua funcionalidade principal é contribuir para a montagem de interfaces de usuário, facilitando sua criação. Como exemplo, o simples arrastar-e-soltar de um item (Field ou DataSet) presente na estrutura do DataSetManager para o Form, faz com que um controle relacionado seja adicionado ao formulário, já devidamente configurado. Tal controle envolvido é totalmente configurável, ficando a cargo do desenvolvedor escolher o seu tipo. TDBGrid, TDBEdit, TDBText, TDBCheckBox e TDBComboBox são alguns exemplos dos tipos disponíveis. Basicamente, a escolha pelo tipo se dá sobre dois contextos: o do conjunto (DataSet) e dos detalhes (Fields). Para o primeiro, leva-se em conta a visualização dos dados do conjunto como um todo, tal qual pelo uso de controles como TDBGrid e TDBCtrlGrid. Já para os "detalhes", estão as configurações do tipo de controle a ser utilizado para cada tipo de dado, quando da utilização de um Field específico.

Ilustrando este cenário, é tomado como base um projeto VCL Forms tradicional que faz uso de um DataSet do tipo TUniTable devidamente ligado à tabela Customer do banco Employee. Isso resulta no carregamento automático do referido elemento para a estrutura DataSet Manager (**Figura 10**).

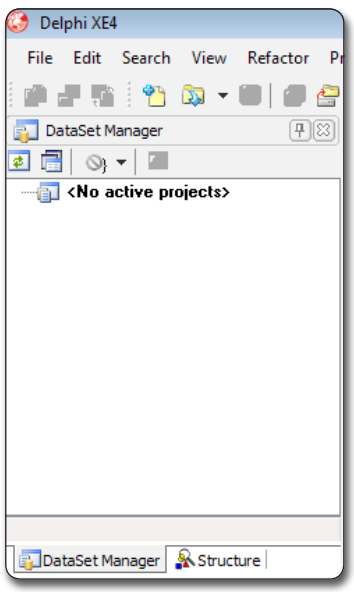

Figura 9. UniDAC – DataSet Manager

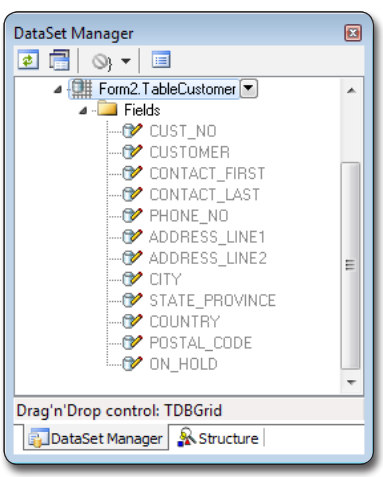

Figura 10. TUniTable Customer na estrutura do DataSet Manager

Após isso, o simples arrastar dos itens internos ao nó Fields para o Form, faz com que sejam adicionados, por padrão, controles TDBEdit já devidamente ligados a cada campo (**Figura 11**), substituindo o processo manual já habitual.

Indo além, conforme mencionado, é possível ainda se alternar entre diversos tipos de controles, tal qual um que abranja o DataSet como um todo (Ex: TDBGrid), conforme mostra a **Figura 12**.

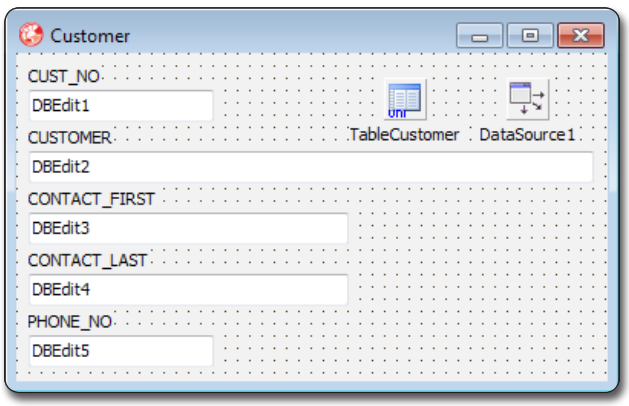

Figura 11. Layout montado a partir do DataSet Manager

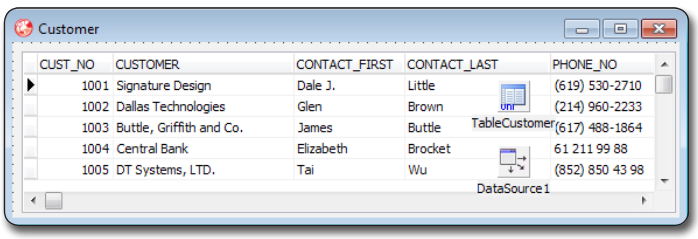

Figura 12. DBGrid gerado a partir do DataSet Manager

Conclusão

Nesta série de dois artigos foi apresentado ao desenvolvedor Delphi mais uma opção de biblioteca de componentes de acesso a dados, o UniDAC, da empresa Devart. Ela surge como uma alternativa comercial, dentre as várias disponíveis no mercado, para tratar de toda a parte de dados de aplicações Delphi. De uma forma geral, opções como esta ganham seu espaço, a partir do momento que proveem recursos e características que vêm a preencher lacunas deixadas pelas soluções nativas. Num exemplo imediato, o desenvolvedor poderia optar pelo uso do UniDAC, em substituição ao dbExpress, em razão de fatores como performance, estabilidade e disponibilidade de recursos específicos a determinado SGBD.

No que diz respeito à parte de acesso a dados da aplicação, seja qual for a opção escolhida em termos de componentes, vale ressaltar a abrangência proporcionada pelo Delphi. Isto porque, são inúmeras as opções de uso, tais quais as já anteriormente citadas (dbExpress, dbGo, IBX, FireDAC e UniDAC), cada qual com sua intenção e recursos particulares. Logo, o que irá firmar a escolha entre uma ou outra opção será, essencialmente, o contexto do negócio a ser atendido pela aplicação que está sendo construída e do próprio desenvolvedor/empresa.

Autor

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ e pós-graduado em Engenharia de Componentes. Atua como Analista Programador Delphi há 8 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas. Certified Delphi Developer. Certified Embarcadero Instructor. Site pessoal: www.fabriciokawata.com.br

Links:

Delphi XE4 - Download Trial
https://downloads.embarcadero.com/free/delphi

Delphi XE4 - Página oficial do produto
http://www.embarcadero.com/products/delphi

UniDAC - Download Trial
http://www.devart.com/unidac/download.html

UniDAC - Página oficial do produto
http://www.devart.com/unidac

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

CURSOS ONLINE

.net magazine

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

MAIS de 25 CURSOS DISPONÍVEIS

CONHEÇA OS CURSOS MAIS RECENTES:

- Curso Padrões de Projeto com C#
- Curso básico de ASP .NET
- Curso de Introdução ao .NET Framework
- Curso Básico de C#
- C# 5 e suas novidades
- ASP.NET MVC – Sistema de Vestibular

Para mais informações :

www.devmedia.com.br/curso/netmagazine

(21) 3382-5038

http://www.devmedia.com.br

DEVMEDIA

FastReport: Criando relatórios para VCL e FMX

Do básico ao avançado

A partir da versão XE2, o FastReport passou a ser a ferramenta de relatórios "oficial" do Delphi, embora ainda seja possível utilizar qualquer outro gerador de relatórios, como o Rave. Com ele podemos criar desde relatórios simples, até os mais complexos e voltados para aplicações Win32 e FireMonkey. Mas existe um porém, a versão que acompanha o Delphi (chamada Embarcadero Edition), não possui suporte ao FireMonkey. Deve ser feito download no site (seção **Links**) para essa versão específica. Os exemplos mostrados neste artigo estão compatíveis com o FireMonkey, aplicando-se as características das plataformas Windows e MAC. O próprio fabricante não disponibiliza documentação específica para desenvolvimento FireMonkey, devido a sua compatibilidade com a VCL, assim, o melhor fazer é utilizar a da documentação para VCL.

A ferramenta ainda possui suporte (adquirido separadamente) para .NET, Mono e Lazarus. Além de uma ferramenta chamada FastCube para análise de dados e construção de relatórios e gráficos (OLAP), entre outras.

Para ver exemplos e outras informações sobre essas ferramentas, veja na seção **Links** os respectivos endereços. Nesse artigo, vamos conhecer a versão do FastReport que acompanha o Delphi XE5, aprendendo a criar relatórios simples, agrupados, formatações, templates e muitos outros exemplos.

Veremos também alguns exemplos para a versão comercial do FastReport, onde há recursos de gráficos, cross-tab entre outros. O FastReport possui conversões de relatórios Quick Report, Rave Reports e Report Builder por meio de units. Este artigo não visa abordar estes exemplos.

Fique por dentro

Este artigo aborda o FastReport, a suíte de geração de relatórios para Delphi. Presente como uma solução a parte há várias versões do Delphi, a mesma passou a ser a incorporada como ferramenta oficial de desenvolvimento de relatórios a partir do Delphi XE2. O tema é útil para o desenvolvedor que deseja conhecer como criar relatórios com o FastReport. Quem já utiliza a ferramenta conhecerá funcionalidades avançadas para serem usadas no dia-a-dia do desenvolvimento de relatórios no Delphi.

Conhecendo a ferramenta

Ao abrir o Delphi e criar um novo projeto VCL Forms, pode-se visualizar a aba FastReport 4.0 (**Figura 1**).

Para a criação de relatórios, é preciso basicamente de apenas dois componentes: o frxReport e o frxDBDataSet. O primeiro é o editor de relatórios, onde basta dar um duplo clique para que o IDE de Design seja aberto (**Figura 2**). O segundo é o responsável por realizar a ligação dos dados com o relatório.

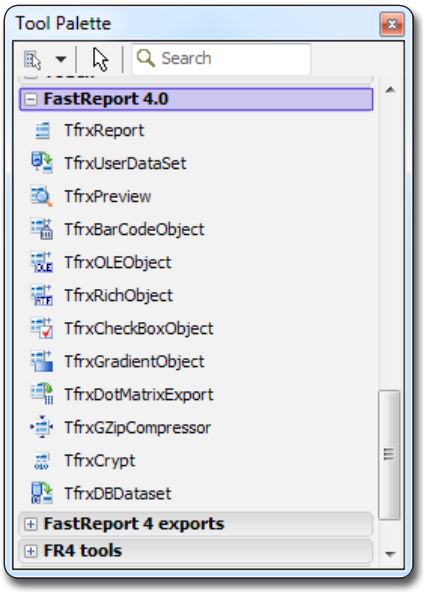

Figura 1. Aba do Fast Report no Delphi

A descrição de cada funcionalidade, segundo a **Figura 2** é:

1. Report designer: onde são adicionados os objetos para o relatório (bandas, controles etc);
2. Barra de menus;
3. Barra de ferramentas: botões com as opções de carregar relatórios, criar novos, salvar etc;

4. Barra de ferramentas do objeto: objetos que podem ser adicionados ao relatório;
5. Report page tabs: abas para configuração da página do relatório;
6. Report tree: janela onde pode-se visualizar as bandas e objetos inseridos no relatório;
7. Object Inspector: janela para definição das propriedades dos objetos do relatório;
8. Data tree: janela com os objetos de dados do relatório. Permite arrastar facilmente um campo para o relatório;
9. Régua;
10. Status line: semelhante a uma barra de status, com informações sobre o objeto selecionado no relatório.

Criando o exemplo

Primeiramente, crie uma aplicação VCL Forms - Win32 (File>New>VCL Forms Application - Delphi). Se você não possui o Delphi XE5, os exemplos aqui vistos poderão ser aplicados a versões anteriores (até o XE2). No Form que se abre, adicione um frxDBDataSet e um frxReport. Crie uma conexão com um banco de dados de sua preferência, como o "EMPLOYEE.fdb" do Firebird. Para este exemplo será usado o SQL Server Express 2012 (ver seção **Links**). Como é uma boa prática separamos o acesso a dados, coloque-os em um Data Module, contendo os componentes DB.

> **Nota**
>
> Criar uma conexão com o banco de dados é bastante simples. Basta escolher a tecnologia que você deseja usar: dbExpress, ADO, IBX etc. No entanto, este assunto não será abordado em maiores detalhes.

Vincule o frxDBDataSet com o DataSet (no Data Module) que retornará os dados da sua consulta. Imagine o frxDBDataSet como um DataSource, que faz a ligação entre controles DataAware e um DataSet. Dê um duplo clique no frxReport para abrir o editor.

Primeiramente, vamos vincular o frxDBDataSet com o relatório, acessando o menu "Report>Data". No editor que se abri, selecione o respectivo controle de dados. Com a vinculação realizada, pode-se iniciar a criação do relatório propriamente dito.

Clique no botão Insert Band na barra de ferramentas de objetos. Na **Figura 3** é mostrado um menu suspenso com todas as opções de banda.

Adicione ao relatório as bandas os tipos: Page Header, Page Footer e Master Data.

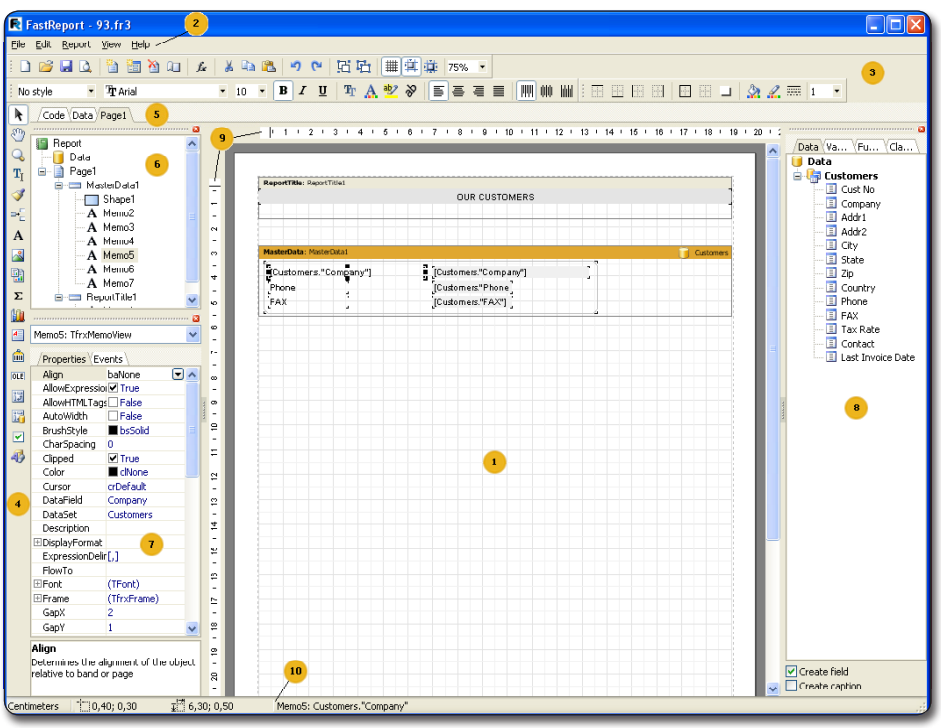

Figura 2. Ambiente de desenvolvimento de relatórios (fonte: FastReport User's Manual)

O Master Data abre um editor solicitando o respectivo DataSet da aba. Selecione então o frxDBDataSet1. Para exibir os dados na banda, existem duas maneiras.

Arraste os campos da janela Data Tree para o relatório e, ao passar o mouse, é mostrada uma seta onde podemos clicar. A partir daí é exibido um menu com os campos da consulta, assim fica fácil modificar o objeto (**Figura 4**).

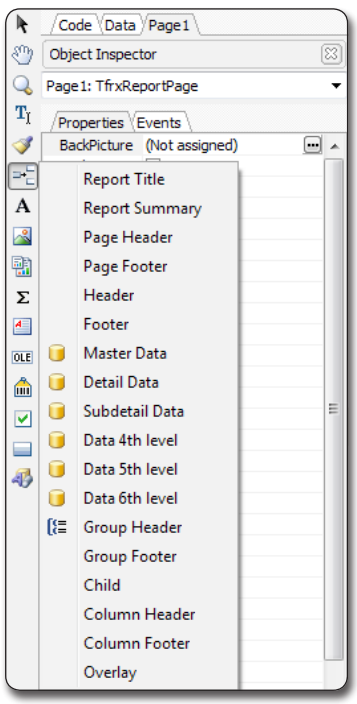

Figura 3. Adicionando bandas ao relatório

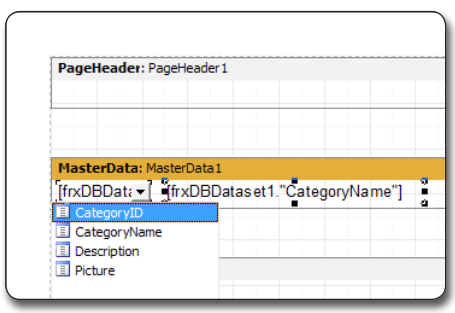

Figura 4. Mudando o campo do objeto facilmente

Outra maneira de adicionar os campos no relatório é escolhendo um Text Object na barra de ferramentas e adicionando o mesmo no relatório. Será aberto um editor com alguns botões (**Figura 5**).

No editor, temos abas para formatações do texto, como fonte, cor, formato de exibição e etc. O primeiro botão da aba Text,

acessa outro editor, onde podemos escolher os campos da consulta, variáveis do relatório, funções, entre outros (**Figura 6**).

Podemos usar o editor do Text Object para apenas adicionar rótulos (textos) no relatório, onde basta digitar o texto desejado na aba Text. Usaremos isso para adicionar o título das colunas, pois as mesmas precisam de texto e não estão vinculados a campos da consulta.

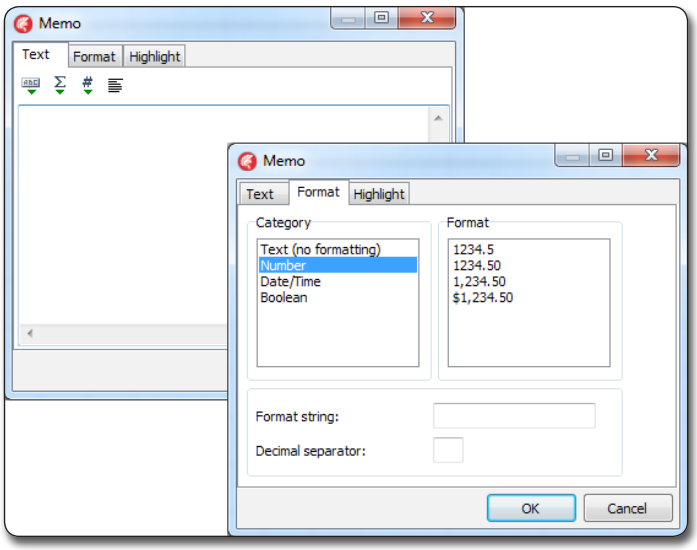

Figura 5. Editor do Text Object

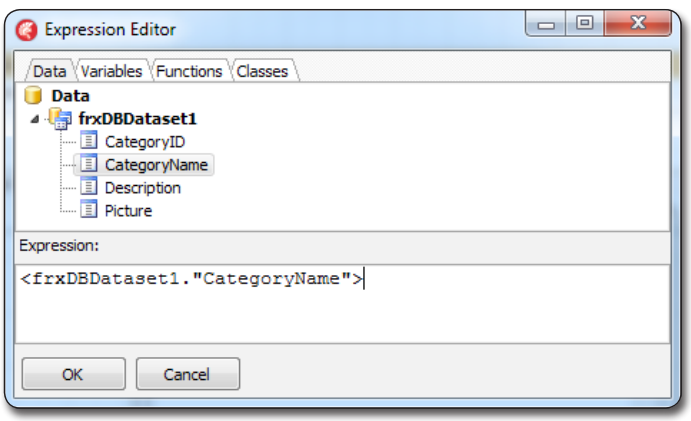

Figura 6. Editor de expressões do relatório

Na barra de ferramentas encontramos as funcionalidades necessárias para formatar os textos digitados no objeto. Ao adicionarmos um Text Object, ou mesmo um campo usando a janela Data Tree, existem linhas visuais que auxiliam no alinhamento, tanto horizontalmente, como verticalmente. Feche o relatório. No Delphi, insira um botão e adicione o seguinte código em seu evento OnClick:

```
frxReport1.ShowReport();
```

Execute a aplicação e temos o primeiro relatório em funcionamento (**Figura 7**).

Formatação de objetos

Em vários casos, temos necessidades de formatação de campos, seja para o tipo data, valores monetários ou outros. Para essa formatação, basta acessar o campo no relatório e usar a aba Format do editor do campo. Crie um novo relatório onde teremos um campo de tipo data e um valorado. No editor, usando a aba Format, configuraremos as opções conforme a **Figura 8,** onde temos o tipo de formatação e o separador decimal (vírgula) do formato (nesse caso monetário).

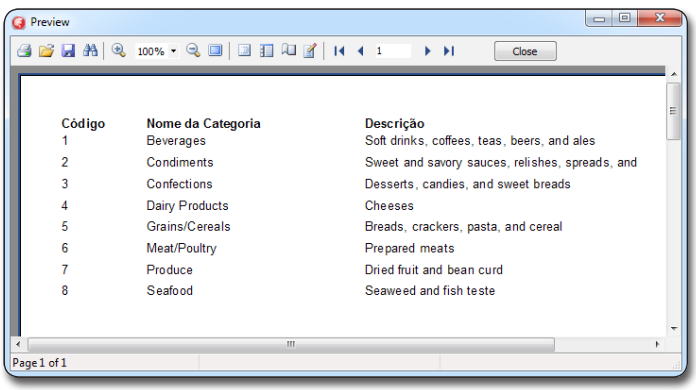

Figura 7. Relatório de listagem em execução

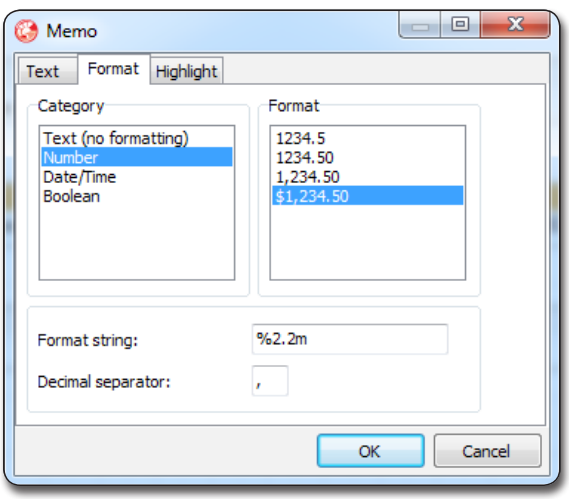

Figura 8. Relatório de listagem em execução

Para o campo data, a funcionalidade é a mesma, onde digita-se o formato desejado ou escolhe os pré-definidos pela ferramenta.

Não é necessário executar o projeto novamente, basta acessar o botão de preview no próprio ambiente do relatório para visualizar as modificações. Podemos usar também a barra ferramentas do IDE para formatações (negrito, itálico e etc), assim como alinhamento do texto nos objetos já inseridos.

E a formatação de um objeto (fonte, cor etc) de acordo com alguma condição? Como por exemplo, em um relatório de estoque de produtos onde o usuário quer visualizar em negrito e vermelho os produtos que estão abaixo do estoque mínimo. É bastante simples! Dê um duplo no campo que deseja implementar, clique

para acessar o editor e a aba Highlight. Na opção Condition, digite: "<frxDBDataset1."TOTAL"> > 500". Marque em Font a opção Bold. Em Color, indique a cor do texto quando a condição do objeto for verdadeira.

Você também pode usar o botão para acessar o editor de expressões para montar a mesma manualmente. Neste exemplo, o campo Total deve aparecer em vermelho quando o valor for maior que 500,00 (**Figura 9**).

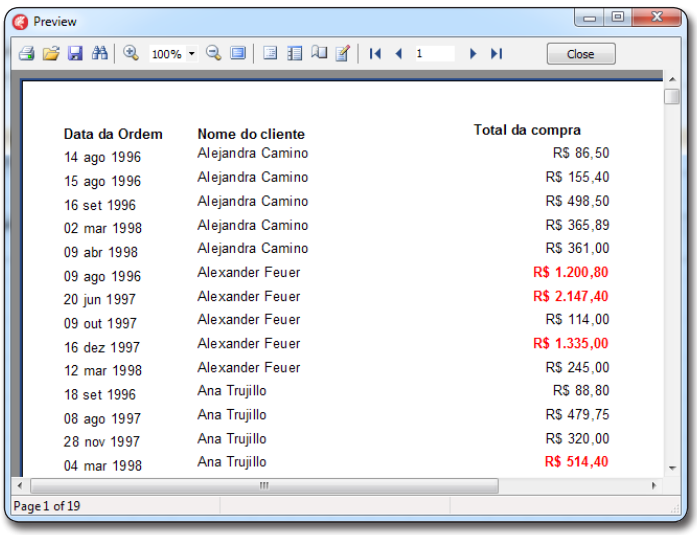

Figura 9. Formatação condicional do relatório

Caso a necessidade seja de mudar a formatação de toda "a linha", basta selecionar todos os objetos e acessar a propriedade Highlight>Condition e fazer a mesma configuração anterior. Outra formatação possível, é referente a quantidade de páginas e a página atual do relatório. O FastReport possui variáveis que retornam informações para serem utilizadas.

Na aba Variables do Data Tree, arraste para o relatório um Date e um TotalPages. Para customizar essas variáveis, basta dar um duplo clique e digitar o texto que desejar. Veja na **Figura 10** o rodapé do relatório.

Figura 10. Formatação do rodapé do relatório

Nota

Para adicionar um texto, basta digitá-lo no editor, tomando o cuidado de não remover a variável.

Nota

Uma característica do FastReport é que ele persiste a formatação de objetos durante a criação de relatórios. Você deve ter notado que após formatar um objeto, ao adicionar outro, ele é mostrado com a mesma formatação do objeto anterior.

Agrupamento

Vamos fazer a mesma listagem do exemplo anterior, com a diferença que dessa vez será realizado o agrupamento pelo nome do cliente. Veja o SQL da consulta na **Listagem 1**.

Listagem 1. Consulta para agrupamento de dados

```
select o.OrderID, o.OrderDate, c.ContactName,
  sum(od.Quantity * od.UnitPrice) as Total
from Orders o
inner join [Order Details] od on od.OrderID = o.OrderID
inner join [Customers] c on c.CustomerID = o.CustomerID
group by o.OrderID, o.OrderDate, c.ContactName
order by c.ContactName
```

Criaremos um novo formulário e adicionar os componentes do FastReport usados anteriormente. Faça as mesmas configurações anteriores (vinculação de dados, bandas etc). Para o agrupamento, usaremos uma banda Group Header, que ao ser adicionada, um editor é aberto para configurarmos o grupo.

Vamos indicar que o agrupamento se dará pelo campo ContactName. Nas opções de agrupamento podemos configurar para que cada grupo seja mostrado em uma nova página ou drill-down, onde clicamos sobre o grupo para mostrar os dados.

Arraste a banda para que fique acima da Master Data. Veja na **Figura 11** o relatório em execução.

Dica

Com a opção drill-down, ao executar o relatório, apenas os grupos serão mostrados. É necessário clicar em cima do mesmo para exibir os registros.

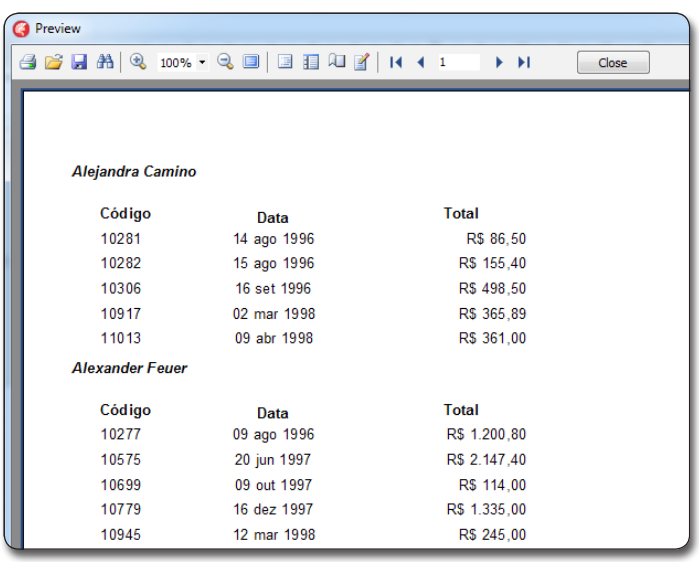

Figura 11. Relatório agrupado

Somatório

Seguidamente, ao usarmos agrupamento, precisamos somar um determinado campo ou contar a quantidade de registros

daquele grupo. Vamos aproveitar esse relatório para fazer esse exemplo. Adicione uma banda GroupFooter no relatório. Em seguida, insira um Text Object e acesse a opção Insert Aggregate da aba Text. No editor, vamos configurar um campo que terá seu valor somado, que no caso é o campo Total. Veja na **Figura 12** como ficou a configuração.

Faça a formatação semelhante ao campo Total da Master Data. Execute o relatório e observe que temos o somatório por grupo. E o valor total no final do relatório? Basta adicionar uma banda ReportSummary e fazer a mesma configuração anterior.

Veja na **Figura 13** o relatório com somatório do grupo e somatório total, depois de inserida a nova banda.

Dica

Caso tenhamos a necessidade de mostrar um somatório acumulado por grupos, basta marcar a opção Running total na janela Insert Agreggate. Assim, a cada final de grupo teríamos um somatório acumulado.

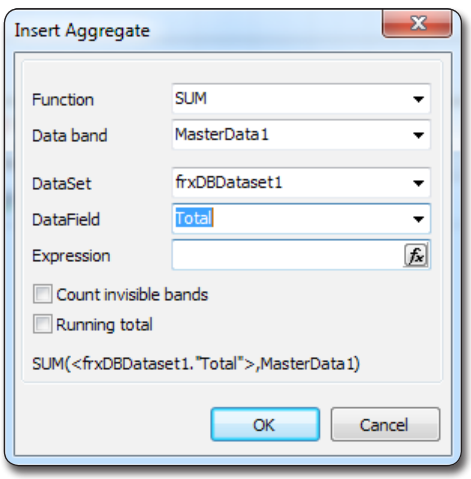

Figura 12. Configurando o somatório do grupo

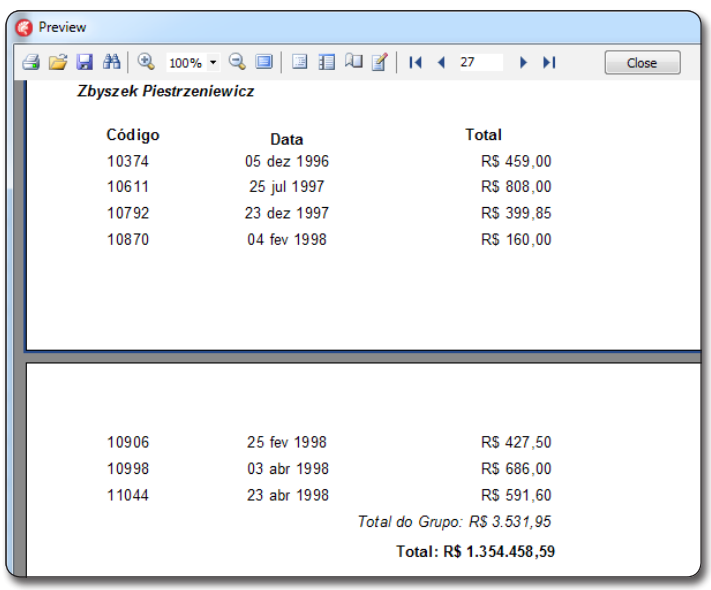

Figura 13. Somatórios no relatório

Master/detail

Outra situação muito recorrente em relatórios é o de master/detail. Exemplo clássico: em uma nota fiscal, os dados referentes ao cliente (comprador) podem ser considerados como master e as informações dos produtos da nota, são o detail, formando uma relação "um para muitos".

Essas duas fontes de dados precisam ter um relacionamento. Primeiro, teremos duas consultas separadas, uma para a master e outra para a detail. Neste exemplo, estamos pesquisando as ordens de compra (Orders) e seus itens (Orders details). Precisaremos de um frxDBDataSet para cada consulta.

O relacionamento será realizado nos componentes de consulta, o que é bastante simples. Adicione um DataSource e faça a ligação com a consulta master. No componente (FireDAC, ClientDataSet, Query, Table etc) com os dados da consulta detail, acesse a propriedade MasterSource e escolha o DataSet anterior. Em MasterFields, acesse o editor e configure o campo de relacionamento (nesse caso OrderID).

Crie um novo relatório e adicione as seguintes bandas: Page Header, Master Data, Detail Data e Page Footer. Na Master Data você deverá configurar o frxDBDataSet que tem os dados do master (no exemplo, Orders) e consequentemente, para o Detail Data, os dados do detail (Orders details).

Veja na **Figura 14** a disposição dos campos nas duas bandas.

As consultas para o master e detail podem ser vistas na **Listagem 2,** onde trouxemos apenas os campos necessários. Execute o relatório e veja que os dados estão agrupados de acordo com o nome do empregado.

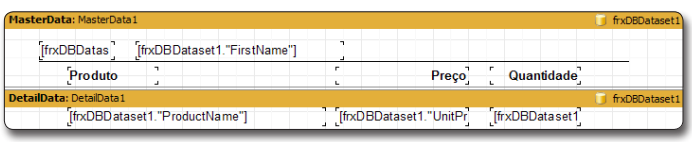

Figura 14. Relatório master/detail

Listagem 2. Consultas para máster e detail do relatório

```
Master
select o.OrderID, o.OrderDate, e.FirstName
from Orders o
inner join Employees e on e.EmployeeID = o.EmployeeID

Detail
select [Order Details].OrderID, [Order Details].UnitPrice,
[Order Details].Quantity, Products.ProductName
from [Order Details]
inner join Products on Products.ProductID = [Order Details].ProductID
```

Exportação

A quantidade de opções de exportação dos relatórios do FastReport é bem grande. Veja na **Figura 15** a aba de exportações do FastReport.

Para exportar, por exemplo, um relatório para PDF, basta adicionar no formulário um componente frxPDFExport. Acesse o IDE de desenvolvimento e execute o preview do relatório.

Note que um botão foi adicionado ao relatório, permitindo agora a exportação.

Para cada tipo de exportação, um item de menu será adicionado no preview (**Figura 16**).

Na versão comercial do FastReport, existe a opção de envio do relatório por e-mail.

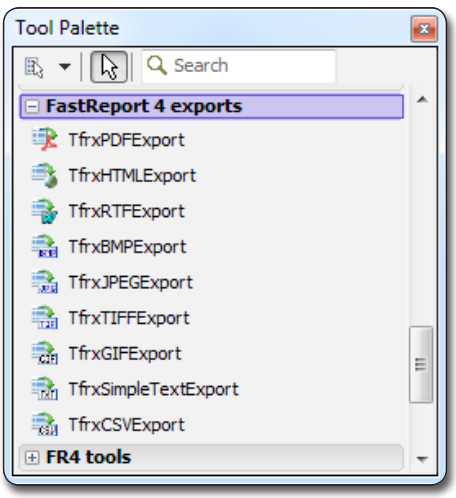

Figura 15. Tipos de exportações do FastReport

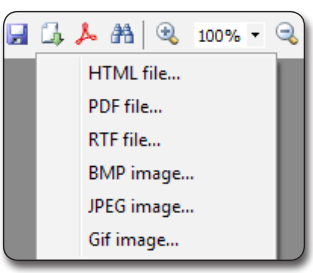

Figura 16. Opções de exportação no preview do relatório

Arquivos de relatórios

Para quem estava acostumado com o Rave Reports, no FastReport cada relatório é um arquivo com a extensão fr3. No Rave, o arquivo RAV refere-se ao projeto e dentro dele, temos um ou vários relatórios. Não é obrigatório salvar para cada relatório um arquivo FR3, pois as configurações, ficam persistidas no frxReport, assim, os relatórios estão embarcados na aplicação Delphi, do modo convencional. Salve um arquivo FR3 se deseja que as atualizações de relatórios não dependam do executável da aplicação. Para carregar o arquivo, usamos o seguinte código:

```
frxReport1.LoadFromFile('Listagem.fr3');
frxReport1.ShowReport();
```

Nota

Se o FastReport não encontrar o arquivo, nenhum erro é exibido. No código Delphi podemos verificar se o arquivo existe e exibir uma mensagem para o usuário.

Usamos o método LoadFromFile para carregar o arquivo de relatório no frxReport. Isso facilita para termos apenas um controle e vários relatórios, sendo carregados de acordo com a necessidade.

Templates

O FastReport oferece a funcionalidade do uso de templates. Com o Quick Report usávamos a herança visual de formulários para facilitar a construção de relatórios com layout padronizado, facilitando a criação e manutenção dos mesmos.

No Rave Reports temos o recurso de Global Pages para "simular" essa funcionalidade. No FastReport usamos templates, onde podemos criar um relatório (arquivo fr3) com o layout padrão e depois usá-lo na criação dos novos relatórios.

No design do FastReport, acesse o menu File>New. Será aberta uma janela com alguns templates pré-definidos (**Figura 17**).

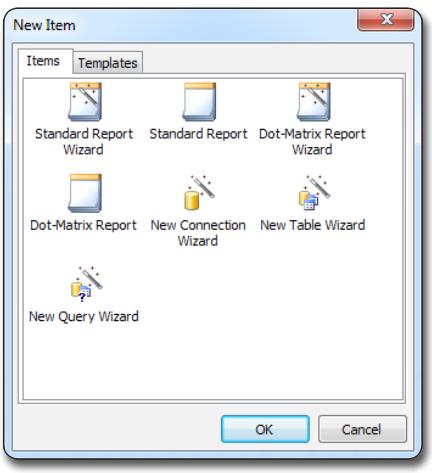

Figura 17. Adicionando um novo item

Na aba Templates, temos alguns modelos que podem ser utilizados. Nesse caso, como nada foi definido, este é mostrado em branco. Crie um novo relatório adicione bandas e controles de tela, como mostrado na **Figura 18**.

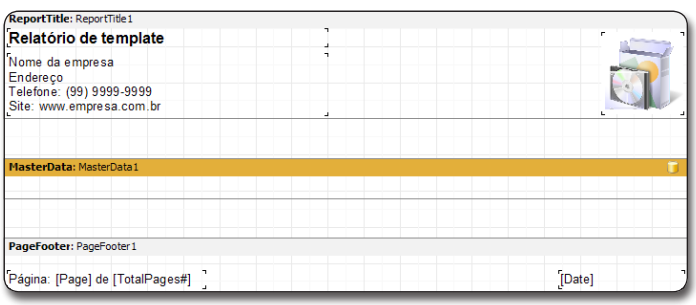

Figura 18. Layout padrão para relatórios do sistema

Nota

Não podemos ter nenhuma vinculação de dados no template. Apenas os relatórios herdados poderão ser ligados à dados.

Esse é o layout que será usado por todos os relatórios. Salve o arquivo fr3 como "LayoutPadrao.fr3" em um diretório. Acesse o menu Report>Options. Na aba Inheritance indicamos o local (diretório) dos arquivos fr3 que serão usados como templates (**Figura 19**).

Agora basta acessar o menu File>New, que o template estará pronto para ser utilizado (**Figura 20**).

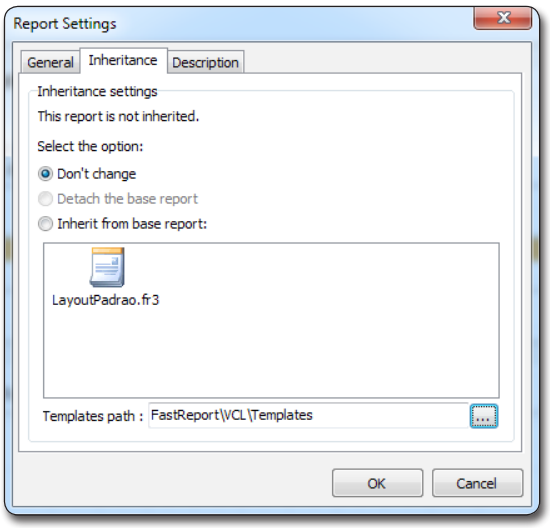

Figura 19. Indicando o diretório onde teremos os templates

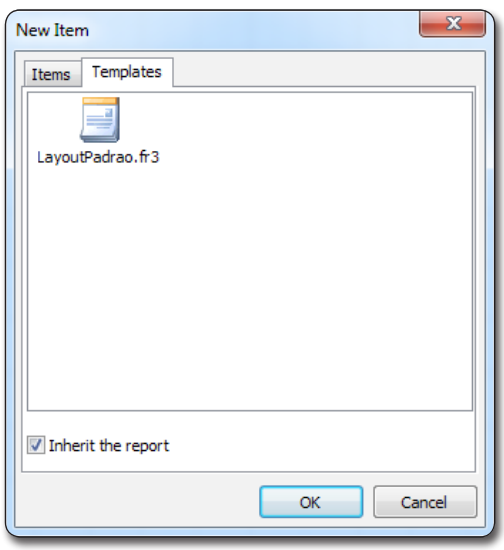

Figura 20. Escolhendo o template criado para um novo relatório

Criando gráficos

Gráficos mostram de forma mais amigável informações do que uma listagem por exemplo. Eles exibem informações de forma visual, facilitando a avaliação dos resultados.

Primeiro faremos a consulta que retorne os dados necessários para a criação do relatório. A consulta, pode ser considerada a parte mais importante do gráfico, pois, precisamos retornar dados que sejam interessantes de serem avaliados em um gráfico.

Um exemplo de gráfico, seria a quantidade de produtos ou funcionários por setor e etc. Neste exemplo é exibido um somatório das vendas do empregados. O SQL usado encontra-se na **Listagem 3**.

Nota

A partir desse ponto, os exemplos precisam de componentes, que estão disponíveis apenas para a versão comercial (full). Baixe a versão trial (já disponível para a versão XE5) no site apresentado na seção **Links**.

Listagem 3. Consulta para o gráfico

```
select Employees.EmployeeID, Employees.FirstName,
SUM([Order Details].Quantity * [Order Details].UnitPrice) as Total
from [Order Details]
inner join Orders on Orders.OrderID = [Order Details].OrderID
inner join Employees on Employees.EmployeeID = Orders.EmployeeID
group by Employees.EmployeeID, Employees.FirstName
```

No retorno da consulta teremos os campos FirstName e Total. Adicione um frxChartObject no formulário, pois, somente com esse controle teremos acesso ao gráfico no FastReport. Adicione os outros controles para criação do relatório, abra o design do FastReport e faça a ligação para a fonte de dados (consulta no Delphi).

Adicione um Chart Object e será aberto seu editor de configuração (**Figura 21**).

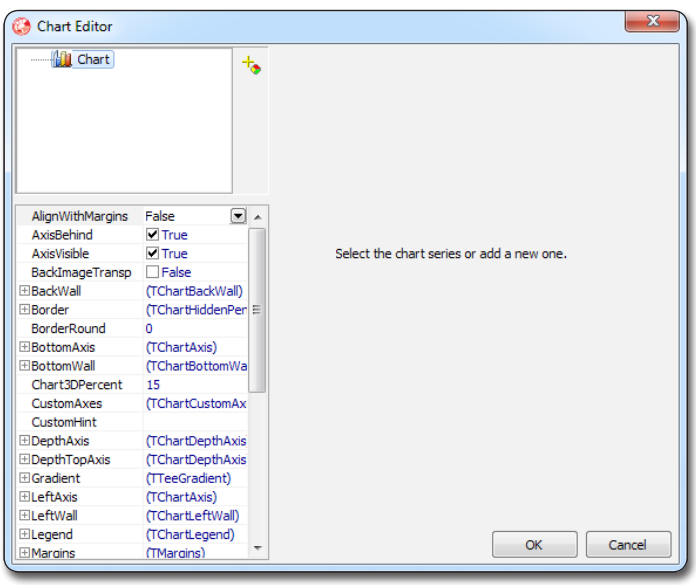

Figura 21. Editor do Chart object

No editor, adicionamos as séries (onde cada uma representa um gráfico), lembrando que podemos ter várias series que podem ser diferentes tipos de gráficos (em barra, pizza, linha etc). Clique no botão Add Series para criamos um novo gráfico.

Na janela que se abre, podemos ver as opções de gráficos e suas derivações (**Figura 22**).

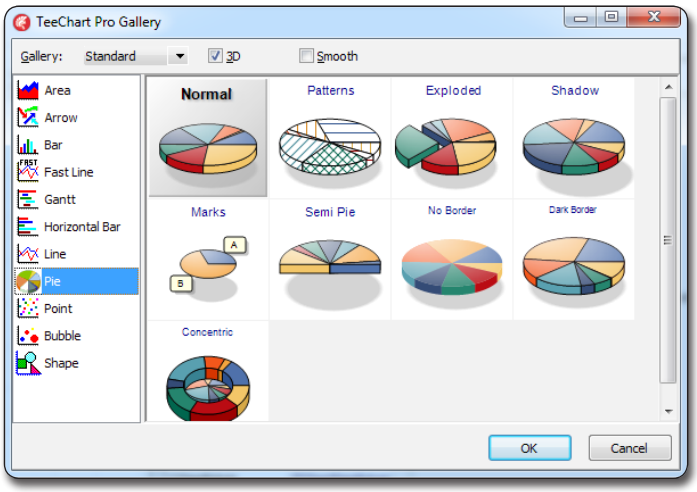

Figura 22. Galeria com os tipos de gráficos disponíveis

Escolha o item Pie e o tipo Normal. Na opção DataSource, selecione o frxDBDataSet adicionado no formulário Delphi. No Label, vamos indicar o campo FirstName da consulta, pois queremos saber quais os empregados e suas vendas.

No item Pie (que muda de nome, de acordo com o tipo de gráfico escolhido), vamos configurar o campo que retorna o somatório de vendas. Caso queira criar gráficos com valores fixos, o que é bastante difícil, basta escolher Fixed data e digitar os valores em Label e Pie.

Pronto, já temos um gráfico pronto para uso. Execute o relatório e veja–o em execução (**Figura 23**).

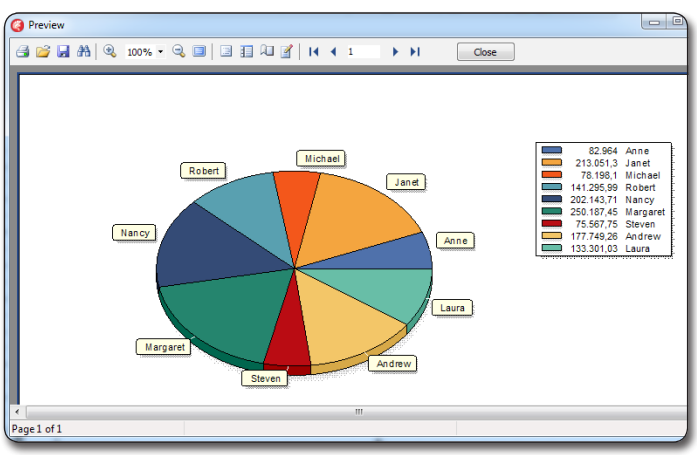

Figura 23. Relatório em execução

Para adicionar um título, selecione Chart no editor. Acesse a propriedade Title>Text. Temos várias configurações para a propriedade Title. Vimos que o gráfico mostra uma quantidade grande de informações e que em alguns casos precisamos apenas das mais relevantes, como por exemplo, os cincos primeiros vendedores no geral.

Para conseguir esse "agrupamento" de informações, é bem fácil. Na série que criamos, temos um GroupBox chamado Other options.

A opção TopN values indica a quantidade de informações que queremos agrupar, por exemplo, para saber os primeiros vendedores, digite "5" no item.

Assim, são mostrados no gráfico, os quatros primeiros vendedores e um item com a soma de todos os outros. Esse último item não retorna nenhum título, mas podemos colocar um texto indicativo no item TopN caption (**Figura 24**).

Veja na **Figura 25** outros tipos de gráficos que podemos criar.

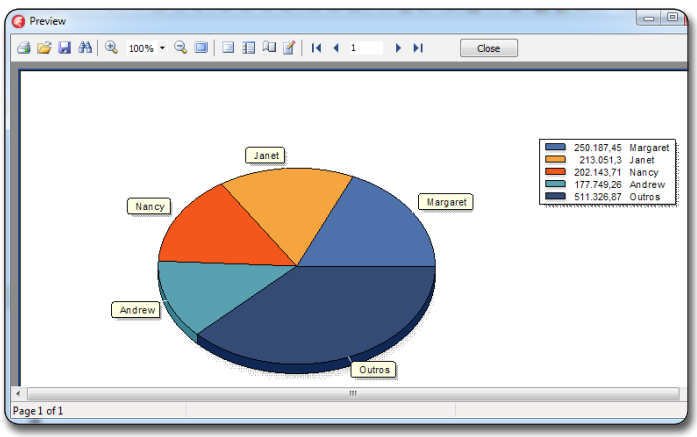

Figura 24. Agrupando informações com o TopN

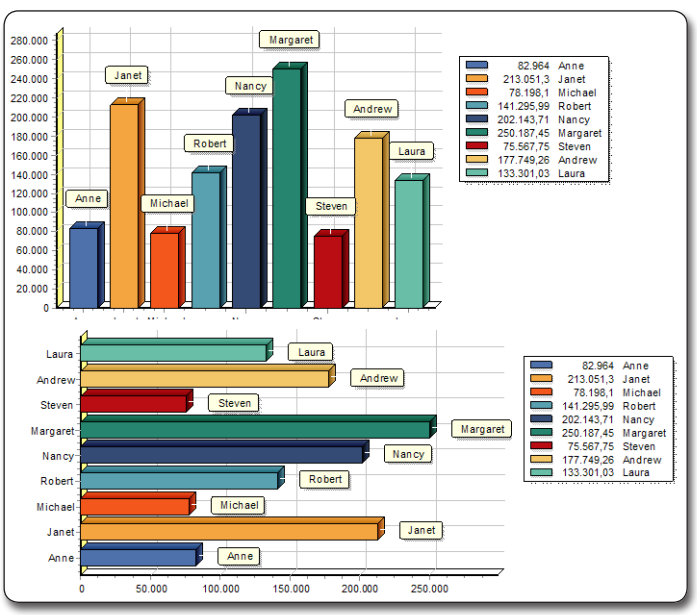

Figura 25. Tipos de gráficos que podemos criar no FastReport

Cross-tab

Cross-tab é um recurso que como o próprio nome sugere, serve para cruzarmos dados de duas ou mais tabelas para que possamos encontrar uma relação que retorne informações para serem apresentadas. Um exemplo simples: montante de vendas do ano, mês ou período por cliente.

Neste caso, as linhas da consulta são dinâmicas (cliente), como de costume, mas as colunas, também precisam ser dinâmicas

(meses, período). Com o cross-tab, fica fácil de fazer esse tipo de relatório.

Nosso SQL precisa retornar o nome do cliente, o montante gasto e o período da venda (nesse caso, será mensal). Veja na **Listagem 4** a consulta.

Note que a consulta retorna o montante gasto pelo cliente agrupado por mês. Faça a ligação da consulta com o relatório, conforme exemplos anteriores. No design, temos o Cross-tab object (onde adicionamos dados manualmente) e o DB Cross-tab object (vinculado a uma fonte de dados).

Adicione um ao relatório e será aberto o editor de configuração, conforme a **Figura 26**.

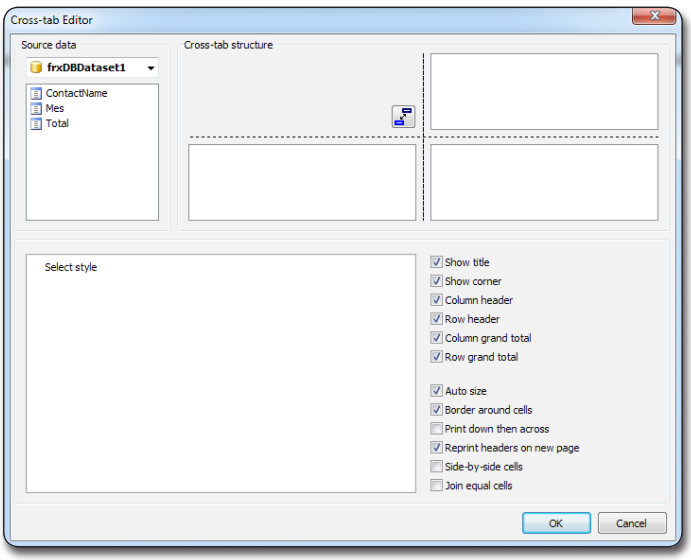

Figura 26. Editor de configuração do DB cross-tab object

Listagem 4. Consulta para croos tab

```
select Customers.ContactName, MONTH(Orders.ShippedDate) as Mes,
SUM([Order Details].Quantity * [Order Details].UnitPrice) as Total
from [Order Details]
inner join Orders on Orders.OrderID = [Order Details].OrderID
inner join Customers on Customers.CustomerID = Orders.CustomerID
where Orders.ShippedDate is not null
group by Customers.ContactName, MONTH(Orders.ShippedDate)
```

Em Source data, escolhemos a fonte de dados (caso a mesma não venha selecionada). No cross-tab structure, configuramos as linhas e colunas. Arraste o campo ContactName para a linha. O campo Mês será a coluna e Total será a parte centralizada.

As linhas e colunas possuem ordenação e agrupamento, já o campo centralizado possui configurações de cálculos, como somatório, contador, média, mínimo e máximo. Na parte inferior do editor, temos um preview de como ficará o relatório.

Ao lado direito, há várias opções de configuração como: mostrar o título, cabeçalho de coluna e linha, totalizadores de colunas e linhas etc. Veja na **Figura 27** o resultado após as configurações.

Basta executar o relatório que teremos um cross-tab (**Figura 28**). Você pode mudar o layout do relatório para paisagem (Orientation = poLandscape) para visualizar todos os dados.

Você pode modificar o SQL para que seja apresentado o nome do mês no lugar do número.

Nota

A formatação mostrada é equivalente a apresentada neste artigo. Selecione as células que deseja formatar, clique com o botão direito e escolha Display Format.

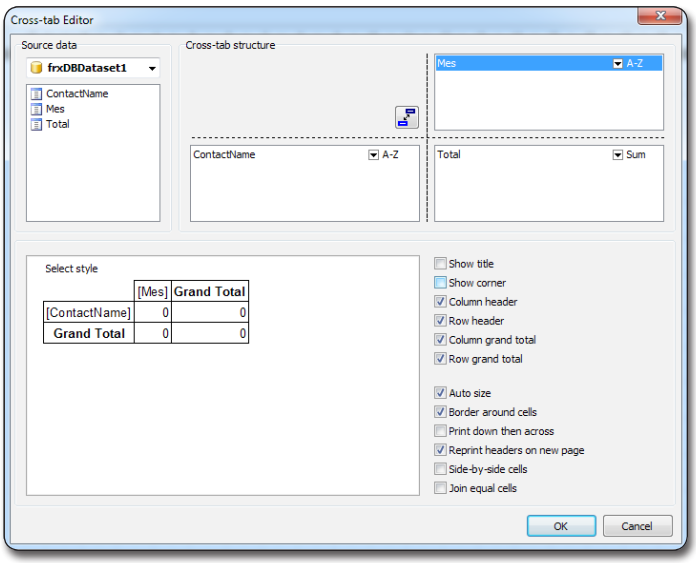

Figura 27. Configuração do cross-tab

	1	2	3	4	5	6	7	8	9	10	11	12
Alejandra Camino			365,89	361,00				241,90	498,50			
Alexander Feuer	1.335,00		245,00			2.147,40		1.200,80		114,00		
Ana Trujillo			514,40					479,75	88,80			320,00
Anabela Domingues	4.401,62			200,00			1.413,00		1.296,00			
André Fonseca	280,00	1.020,00	155,00		342,00				331,78	801,10	4.922,35	850,00
Ann Devon	3.063,00			4.738,81	5.626,85							1.605,00
Annette Roulet	2.692,15	894,00	936,00	1.226,80				461,00			452,00	3.565,40
Antonio Moreno		660,00		881,25	2.156,50	2.082,00			956,90	375,50		403,20
Aria Cruz	231,40			618,00			387,50		1.820,20		1.215,80	166,00
Art Braunschweiger	678,00	485,00		439,00				48,00		4.819,40	141,60	5.878,70
Bernardo Batista	1.666,65			1.353,60			1.255,38		1.188,80		550,00	959,20
Carine Schmitt		519,00		1.733,06					920,10			
Carlos González	3.107,50	144,00	2.051,60	3.392,00	1.770,00			1.414,80	1.050,60	1.940,00		2.192,00
Carlos Hernández	5.211,28	1.453,50	1.945,10	6.198,80	1.727,50	880,50	3.404,90	2.412,00			378,00	
Catherine Dewey		2.950,50		1.796,08	946,00			1.434,00				3.304,00
Christina Berglund	1.254,00	5.299,15	729,50	2.034,50	3.192,65	1.565,65		2.992,40	5.492,40	630,00	1.459,00	2.318,90
Daniel Tonini		649,00	543,05	800,00								
Dominique Perrier		108,50	2.052,50	210,00							52,35	
Eduardo Saavedra		408,20	137,50				155,00					136,00
Elizabeth Brown	931,50	631,60	156,00									
Elizabeth Lincoln	2.010,50	2.523,00	8.208,60	3.605,30							3.118,00	1.832,80
Felipe Izquierdo	3.836,50	264,00	4.784,90	270,20					2.720,05	1.073,90	1.850,00	
Fran Wilson		134,40		1.575,00	417,20				1.844,00	288,00		

Figura 28. Croos-tab em execução

Código de barras

Atualmente, outro componente muito importante é o barcode. No FastReport podemos configurar o Barcode object de maneira fácil e rápida. Se nossa necessidade for gerar um código de barras padrão, com um único número, basta adicionar o controle no relatório e configurar o seu editor (**Figura 29**).

Indicamos o valor em Code e configuramos o tipo do código de barras (existem muitas opções). É possível também rotacionar o campo no relatório. Caso tenhamos uma listagem e a necessidade

neste caso, obviamente, é de um código de barras por item, basta usar a mesma configuração. Para isso informamos na propriedade Source o campo da consulta.

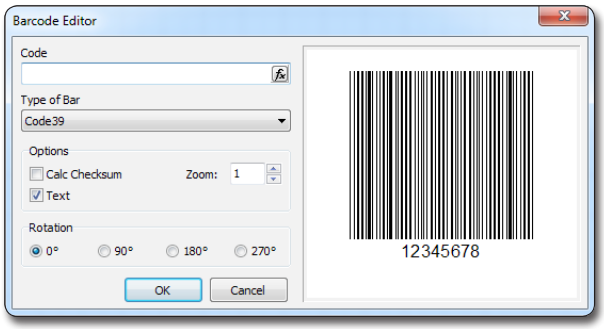

Figura 29. Editor de configuração do Barcode object

Conclusão

Vimos nesse artigo a "nova" ferramenta para gerar relatórios no Delphi. Aprendemos a criar relatórios simples, listagens, o "comum" do dia-a-dia do desenvolvimento de relatórios com Delphi. Além disso, mostramos dicas para gerar relatórios avançados no FastReport, onde foi possível criar gráficos, cross-tabs e outras configurações avançadas.

Existem muitas outras funcionalidades que podem e devem ser explorados pelo leitor para criar relatórios cada vez mais elaborados de acordo com o gosto e a necessidade do usuário.

Autor

Paulo Quicoli
pauloquicoli@gmail.com
Editor da revista ClubeDelphi. Formado em processamento de dados pela FATEC-TQ. Atua como analista programador de sistemas da Siplan Control-M (www.siplancontrolm.com.br) unidade Jaboticabal, e professor da FATEC, campus Taquaritinga.
Blog: http://quicoli.worpress.com

Links:

Página do FastReport
http://www.fast-report.com/pt/

SQL Server 2012 Express
http://www.microsoft.com/pt-br/download/details.aspx?id=29062

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Novidades do Firebird 3

Prepare-se para as mudanças conhecendo a versão Alpha

O Firebird é conhecido da comunidade há pelo menos dez anos e seu desenvolvimento é acompanhado de perto por seus utilizadores. Talvez não seja notório para alguns, mas em seu time de desenvolvimento temos um brasileiro participante.

De todas as versões planejadas, a versão 3 tem sido a mais esperada por todos, pois traz mudanças significativas ao núcleo do Firebird, destacando-se a adequação total para ambientes multiprocessados e máquinas de múltiplos processadores/núcleos. Outras novidades foram incluídas, como o esperado suporte para o tipo de dado Boolean.

Instalação e Gerenciamento

Como a versão Alpha é uma versão de testes, vamos fazer o download da versão Firebird-3.0.0.30566-0_Win32_Alpha1.zip (ver seção **Links**), que é o pacote para instalação manual. Uma vez feito o download, descompactamos para uma pasta de fácil acesso, por exemplo, C:\Firebird3. O objetivo é executar o Firebird como aplicativo e não como serviço.

A primeira mudança percebida é que não existe mais o executável fbserver.exe e sim o firebird.exe, que unifica as arquiteturas em um único arquivo. Para executá-lo como aplicativo utilizamos o comando a seguir no prompt de comando:

```
C:\Firebird3>firebird -a -m
```

Ao concluir a execução, temos na área de notificação do Windows um ícone do Firebird. Ao efetuarmos um clique duplo vemos a tela da **Figura 1.**

Mas atenção, por padrão a porta 3050 ainda é utilizada, e caso já exista uma instância do Firebird sendo executada e configurada para essa porta, haverá um conflito no momento de sua utilização. É possível utilizar o arquivo firebird.conf, localizado na pasta onde o Firebird 3 foi descompactado, para mudar a porta a ser usada pela versão 3. Ao abrir esse arquivo em um editor de texto simples, por exemplo, o bloco de notas, localizamos o parâmetro RemoteServicePort, como mostra a **Figura 2**.

Fique por dentro

Este artigo apresenta uma introdução às mudanças na linguagem SQL do Firebird 3 Alpha. É demonstrada sua instalação, configuração e uso em um ambiente que não irá conflitar com versões previamente instaladas. O intuito é tornar conhecidos recursos novos e já pensar em como será a atualização para nessa nova versão do Firebird, claro, quando for liberação a versão final.

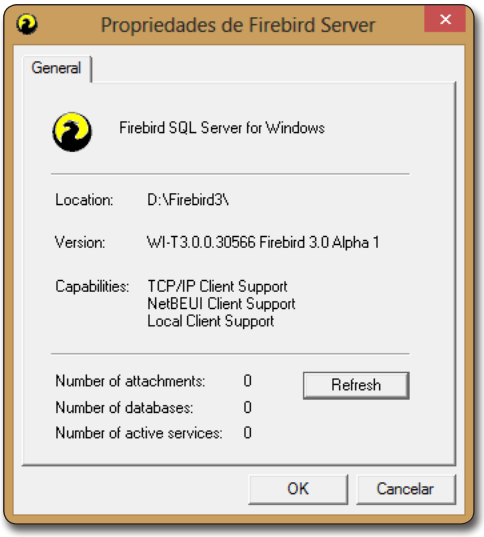

Figura 1. Firebird 3 em execução como aplicativo

```
571
572
573 # ----------------------------
574 # TCP Protocol Settings
575 #
576 # The TCP Service name/Port number to be used for client database
577 # connections.
578 #
579 # It is only necessary to change one of the entries, not both.  The
580 # order of precendence is the 'RemoteServiceName' (if an entry is
581 # found in the 'services.' file) then the 'RemoteServicePort'.
582 #
583 # Per-connection configurable.
584 #
585 # Type: string, integer
586 #
587 #RemoteServiceName = gds_db
588 #RemoteServicePort = 3050
589
590 #
```

Figura 2. Configuração da porta de execução

O sinal # no início de cada linha indica que a mesma está inutilizada, sendo considerada um comentário. Então para mudar a porta para, por exemplo, 3053, removemos esse sinal #, mudamos o número, salvamos o arquivo e executamos novamente o Firebird 3.

Nota

Nesse momento o Windows pode apresentar um diálogo solicitando sua confirmação de liberação de uso de porta TCP/IP, caso isso ocorra, autorize a liberação

Como a versão 3 é Alpha, nem todos produtos de gerenciamento oferecem suporte, contudo, o IBExpert em sua última versão já oferece. Vamos utilizá-lo para explorar as novidades da linguagem SQL. O processo de download (ver seção **Links**) sofreu algumas modificações, em sua página é necessário realizar um cadastro acessado pelo botão Register. É muito importante informar um e-mail válido, pois nele serão enviados o username e password a utilizar para login na página de download, que então disponibilizará um botão para a versão personal do IBExpert. Após seu download e instalação, a tela da **Figura 3** é apresentada.

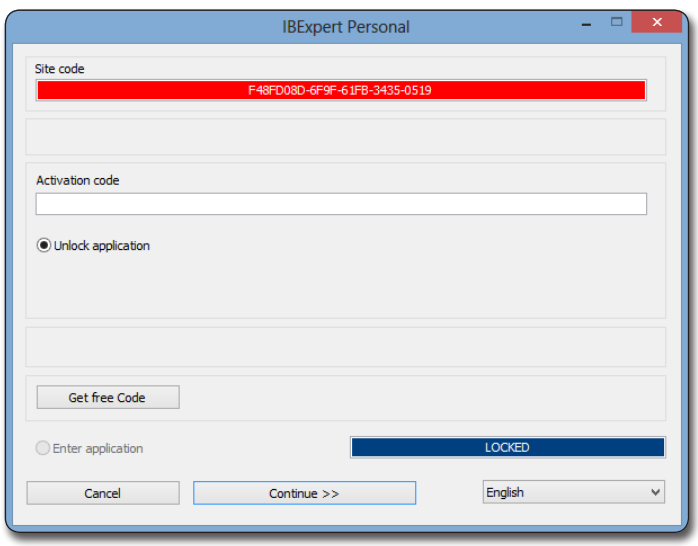

Figura 3. Validação da instalação

É necessário validar a instalação através de um código de ativação que é gerado ao clicar no botão Get free Code. Uma página web é aberta, onde informamos o login e password recebidos por e-mail para a geração do código de ativação. Esse código de ativação é baseado no endereço MAC da placa de rede do computador utilizado, restringindo o uso apenas para o computador instalado. Caso seja necessário instalar a versão Personal em outra máquina, outro cadastro deve ser realizado. De posse do código, basta copiá-lo para o referido campo da **Figura 3** e continuar com a instalação.

Criando um banco de dados

Dentre os arquivos de instalação do Firebird 3 temos o isql .exe, que é uma ferramenta de linha de comando responsável por executar comandos SQL. Vamos utilizá-lo para criar um banco de dados local. Via prompt de comando acessamos a pasta onde o Firebird 3 foi descompactado e executamos o comando isql .exe. Sua execução cria um prompt (SQL>) para entrada de comandos SQL. A seguir criamos o banco de dados com a seguinte instrução:

```
CREATE DATABASE D:\TESTEFB3\TESTE.FDB;
```

Para finalizar o prompt digitamos: exit;

Com o banco de dados criado, podemos agora nos conectar pelo IBEXPERT, como mostra a **Figura 4**.

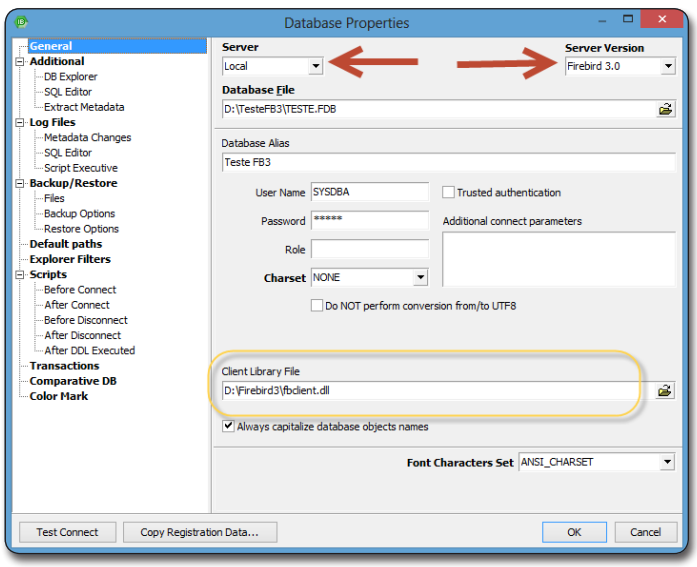

Figura 4. Conectando ao banco de dados existente

Temos alguns pontos destacados. O primeiro é quanto a propriedade Server. Configuramos para Local, para usar a versão embarcada do Firebird, que não requer usuário ou senha. Em Server Version obviamente escolhemos a versão 3 e em Client Library File apontamos para a DLL cliente específica dessa versão. Assim, é possível conectar no banco de dados e iniciar os testes de SQL. O Firebird 3 traz novidades no SQL DDL e no DML. DDL é a categoria de sentença SQL responsável pela Data Definition, ou definição de dados. Com ela criamos as tabelas, campos, os objetos de um banco de dados. DML significa Linguagem de Manipulação de Dados, nela concentra-se comandos como INSERT, UPDATE, DELETE e outros.

Tipo BOOLEAN

Uma das novidades trazidas é a existência do tipo BOOLEAN. Ele é utilizado para armazenar valores lógicos TRUE ou FALSE, e pode aceitar um valor NULL como os outros tipos e um outro específico, que é o valor UNKNOWN (desconhecido).

Para testá-lo criamos uma tabela chamada Cliente e nela inserimos alguns valores, como mostra o script SQL da **Listagem 1**.

O script foi digitado no IBExpert, através da opção Script Executive.

O tipo BOOLEAN não requer explicitamente os valores TRUE ou FALSE, esse valor lógico pode ser gerado a partir de uma expressão, reescrevendo a linha 7, por exemplo:

```
insert into cliente values (2,'CARLA', 4 > 3);
```

A expressão "4 > 3" exibida no código anterior é avaliada e seu valor lógico é inserido no referido campo. Uma vez que a tabela está preenchida, podemos utilizar esse novo tipo para realizar consultas, como mostra a **Listagem 2**.

Podemos observar na linha 3 que não foi informado o valor para o campo ativo. Automaticamente é assumido o valor TRUE. A **Figura 5** exibe o resultado no IBExpert.

Se for necessário pesquisar pelos clientes não ativos, a mesma consulta poderia ser reescrita de duas formas, como mostra a **Listagem 3**.

Nas linhas 3 e 7 vemos duas formas diferentes de dizer que queremos os registros cujo campo ativo seja falso. O resulta de ambas as consultas é o mesmo, apenas o registro 2 é retornado. Porém existe uma terceira forma, exibida na **Listagem 4**, que traz o resultado um pouco diferente, como mostra a **Figura 6**.

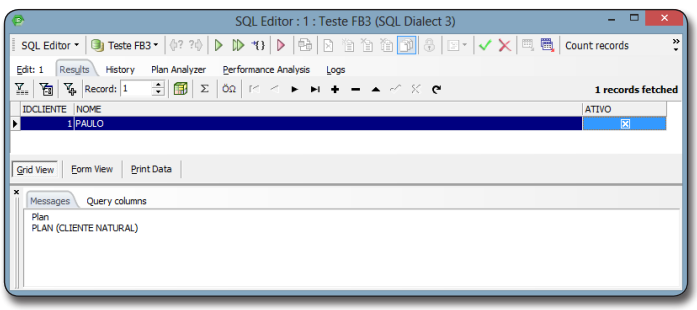

Figura 5. Resultado de uma consulta por campo BOOLEAN

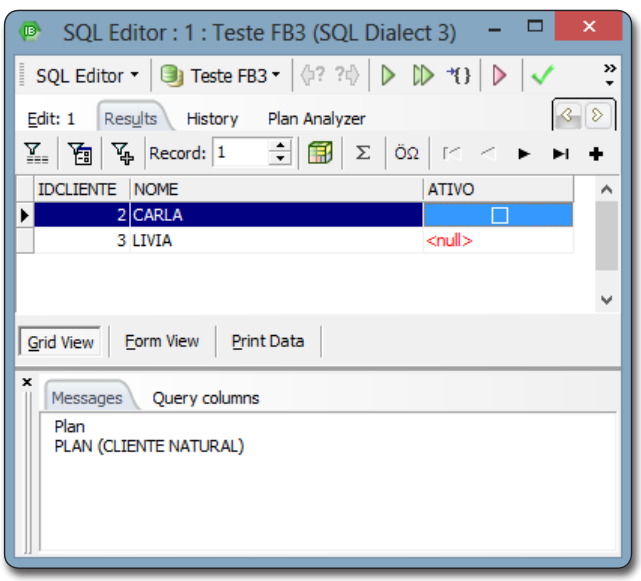

Figura 6. Resultado de pesquisa da **Listagem 4**

Na linha 3 foi utilizado o comando "is not true", que também pode ser utilizando com sua contraparte "is not false". Essa sentença trouxe mais um resultado porque justamente o registro 3, não possui um valor definido para o campo ativo, ou seja, ele não é TRUE mas **não é FALSE**. Para esse tipo de comparação é preciso atenção a esses detalhes. Se for necessário recuperar então o que não é TRUE e nem ao menos FALSE, podemos utilizar duas formas, mostradas na **Listagem 5**.

Listagem 1. Criando uma tabela

```
01 create table cliente(
02 idCliente int,
03 nome varchar(100),
04 ativo boolean);
05
06 insert into cliente values (1, 'PAULO', true);
07 insert into cliente values (2, 'CARLA', false);
08 insert into cliente values (3, 'LIVIA',null);
```

Listagem 2. Recuperando os clientes ativos

```
01 select * from cliente
02 where
03 ativo
```

Listagem 3. Recuperando clientes não ativos

```
01 select * from cliente
02 where
03 ativo = false
04
05 select * from cliente
06 where
07 ativo is false
```

Listagem 4. Buscando valores diferentes de True

```
01 select * from cliente
02 where
03 ativo is not true
```

Listagem 5. Recuperando valores diferentes de TRUE e FALSE

```
01 select * from cliente
02 where
03 ativo is null
04
05 select * from cliente
06 where
07 ativo is unknown
```

Nas linhas 3 e 7 fica claro que podemos, no caso do tipo BOOLEAN, utilizar tanto null quanto unknown para representar valores diferentes de TRUE e FALSE.

Tipo Identity

Até a versão 2.5.X, uma das possíveis soluções para criação de um campo auto incremento que será chave primária, é a utilização do uso de um generator (ou sequence como também é conhecido) e uma trigger para sequenciar esse generator no momento da inserção. Essa rotina é tão constante que o próprio IBExpert automatiza esse processo para nós quando definimos em seu

designer de criação de tabela que determinado campo será uma chave primária com auto incremento.

No Firebird 3 tipo Identity basicamente define que um determinado campo inteiro terá seu valor incrementado automaticamente quando esse não constar em uma cláusula INSERT. Internamente é criado um generator para esse campo que é então gerenciado pelo banco de dados. Contudo, sua unicidade não é garantida por si só, ou seja, simplesmente criar um campo IDENTITY não garante que seu valor não repetirá, para isso é necessário incluir uma restrição UNIQUE (campo único) ou de PRIMARY KEY (chave primária). Para demonstrar seu uso foi criada uma nova tabela, Produto, com alguns valores inseridos, como mostra a **Listagem 6**.

Na linha 2 vemos a sintaxe do tipo identity, que tem por base:

```
<CAMPO> INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY
```

Listagem 6. Tabela Produto com campo Identity

```
01 create table produto(
02 idproduto integer generated by default as identity primary key,
03 descricao varchar(100));
04
05 insert into produto(descricao) values('PRODUTO 1');
06 insert into produto(descricao) values('PRODUTO 2');
07 insert into produto(descricao) values('PRODUTO 3');
```

Ao analisar bem o que foi escrito percebemos que criamos uma chave primária mas não informamos em lugar algum que esse campo é NOT NULL, que é um exigência para chaves primárias. Isso foi possível porque implicitamente todo campo IDENTITY é NOT NULL. Ao verificarmos o resultado da inserção de dados, vemos que claramente é gerada uma sequência (**Figura 7**), desde que o campo não faça parte do comando INSERT.

Se fosse necessário alterar a tabela Cliente, que já possui valores, para ter seu campo IDCLIENTE como uma chave primário de auto incremento, não seria possível porque isso não é permitido. A solução seria criar um novo campo que seria a chave primária Identity, e transferir para ela os valores do campo IDCLIENTE atual.

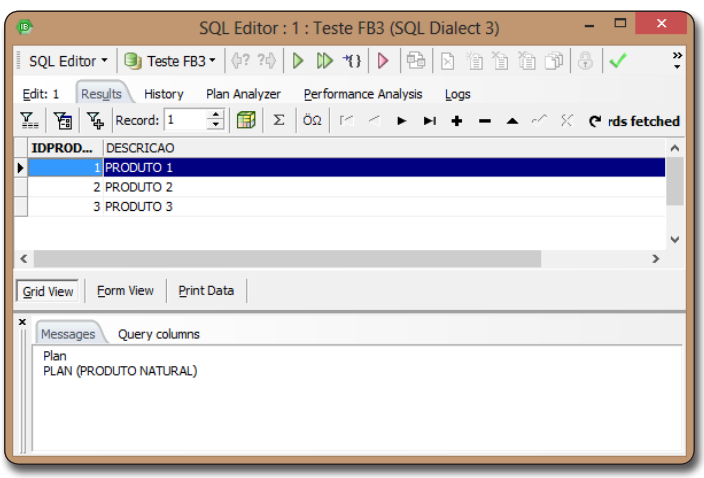

Figura 7. Campo com valores gerados automaticamente

Comando MERGE

Basicamente o MERGE é utilizado para ler informações de uma tabela e inserir ou atualizar essas informações em outra, baseando-se em uma condição. Para exemplificar vamos abordar um cenário de reestruturação de banco de dados. Digamos que em nosso banco de exemplo temos uma tabela ProdutoEstrangeiro, como mostra a **Listagem 7**.

Essa tabela não será mais utilizada e seu conteúdo precisa ser transferido para a tabela Produto, já existente. É necessário manter o conteúdo da tabela Produto e caso algum registro em ProdutoEstrangeiro já exista em Produto, nada deve ser feito, mas se o registro não existe, deverá ser incluído em Produto.

O comando MERGE permite perfeitamente essa execução, como mostra a **Listagem 8**.

Listagem 7. Tabela ProdutoEstrangeiro

```
01 create table produtoestrangeiro(
02 idproduto integer generated by default as identity primary key,
03 descricao varchar(100));
04
05 insert into produtoestrangeiro(descricao) values('PRODUTO 1');
06 insert into produtoestrangeiro(descricao) values('PRODUTO 2');
07 insert into produtoestrangeiro(descricao) values('PRODUTO 3');
08 insert into produtoestrangeiro(descricao) values('PRODUTO 4');
09 insert into produtoestrangeiro(descricao) values('PRODUTO 5');
10 insert into produtoestrangeiro(descricao) values('PRODUTO 6');
```

Listagem 8. Execução do MERGE

```
01 MERGE INTO produto p
02  USING
03    (SELECT * FROM produtoestrangeiro) pe
04    ON (p.idproduto = pe.idproduto)
05  WHEN NOT MATCHED THEN
06    INSERT (idproduto, descricao)
07    VALUES (pe.idproduto, pe.descricao)
```

Um join do tipo right join é realizado entre a tabela produto e produtoestrangeiro. Na linha 3 foi criada uma tabela temporária, que contém os lados do lado "right" do join, ou seja, os dados que serão comparados com a tabela definida na cláusula INTO. Caso o registro de uma não combine com o registro da outra, o INSERT da linha 6 e 7 é executado. Essa verificação de combinação é determinada na linha 4, na cláusula ON. Se o campo idproduto da tabela produtoestrangeiro não existir em produto, esse registro será criado em produto.

Mais situações de combinação poderiam ser feitas, como por exemplo, caso os valores combinem, atualiza-se a descrição em Produto. A **Listagem 9** mostra essa variação.

As linhas 5 e 6 mostram o update caso encontre-se registros com o mesmo id.

Funções analíticas

Também são conhecidas como Window Functions. São funções de agregação que não restringem seu resultado ao conjunto de informações gerados pela cláusula where. Para uma exemplo que esclareça, vamos alterar a tabela Produto para que tenha uma

nova coluna valor e todos os produtos nela existentes possuam esse valor, como mostra a **Listagem 10**.

O script é bem simples, incluímos um campo na tabela produto e preenchemos esse campo com um valor determinado.

É necessário contudo, criar uma sentença SQL que traga cada produto e mostre o quanto o seu valor corresponde do todo, em porcentagem. A abordagem tradicional seria a utilização de uma subquery, como mostra a **Listagem 11**.

Listagem 9. MERGE com mais de uma situação

```
01 MERGE INTO produto p
02  USING
03    (SELECT * FROM produtoestrangeiro) pe
04     ON (p.idproduto = pe.idproduto)
05  WHEN MATCHED THEN
06    UPDATE SET DESCRICAO = PE.DESCRICAO
07  WHEN NOT MATCHED THEN
08    INSERT (idproduto, descricao)
09    VALUES (pe.idproduto, pe.descricao)
```

Listagem 10. Alterando a tabela Produto

```
01 alter table produto
02 add valor numeric(15,2);
03
04 commit;
05
06 update produto set valor = 1200 where idproduto = 1;
07 update produto set valor = 850.30 where idproduto = 2;
08 update produto set valor = 120 where idproduto = 3;
09 update produto set valor = 87.45 where idproduto = 4;
10 update produto set valor = 650 where idproduto = 5;
11 update produto set valor = 35 where idproduto = 6;
```

Listagem 11. Resolução tradicional

```
01 select
02    idproduto,
03    descricao,
04    valor,
05    ((valor / (select sum(valor) from produto))* 100) percentual
06  from produto
07  order by idproduto;
```

Na linha 5 vemos o uso da subquery para se obter o somatório dos produtos e então utilizar esse valor para geração do campo calculado "percentual". Ao olharmos para essa sentença não vemos nada errado, contudo, na prática ela possui um problema de performance. Para cada registro da tabela produto, a subquery será executada. Quando se tem vinte, trinta, cem registros, isso não parece muito. Mas levemos em conta que podemos ter um milhão de produtos lançados. Isso não teria um bom desempenho mesmo.

A solução é utilizar a função de agregação SUM como uma função analítica. Isso é demonstrado na **Listagem 12**.

A diferença é vista na linha 5. Substituímos completamente a subquery pela função SUM seguida da cláusula OVER(). Essa cláusula indica que a função será executada como uma Window Function. Isso faz com que o valor do SUM seja calculado apenas uma vez e não mais a cada registro, gerando um ganho de performance considerável.

Existem outras Window Functions, como a LEAD. Ela é classificada como navegacional, isso porque ela pode exibir o valor do próximo registro em relação ao que está sendo retornado. Pode parecer um pouco confuso isso, mas para demonstrar sua riqueza, incluímos na tabela cliente alguns novos valores, como mostra a **Listagem 13**.

Temos mais seis instruções INSERT gerando novos registros. A sentença que irá demonstrar a utilidade da função LEAD está na **Listagem 14.**

Listagem 12. Utilizando SUM como Window Function

```
01  select
02    idproduto,
03    descricao,
04    valor,
05    ((valor / sum(valor) OVER ())* 100) percentual
06  from produto
07  order by idproduto;
```

Listagem 13. Inclusão de novos registros

```
INSERT INTO CLIENTE (IDCLIENTE, NOME, ATIVO)
  VALUES (4, 'PAULO ROBERTO', TRUE);
INSERT INTO CLIENTE (IDCLIENTE, NOME, ATIVO)
  VALUES (5, 'PAULO JOSE', TRUE);
INSERT INTO CLIENTE (IDCLIENTE, NOME, ATIVO)
  VALUES (6, 'CARLOS DA COSTA', TRUE);
INSERT INTO CLIENTE (IDCLIENTE, NOME, ATIVO)
  VALUES (7, 'CICERO DA SILVA', TRUE);
INSERT INTO CLIENTE (IDCLIENTE, NOME, ATIVO)
  VALUES (8, 'LEONARDO ALVES', TRUE);
INSERT INTO CLIENTE (IDCLIENTE, NOME, ATIVO)
  VALUES (9, 'LEANDRA CUNHA', TRUE);
```

Listagem 14. Empregando a função LEAD

```
01 select
02   nome,
03   lead(nome) over (order by nome) proximo
04 from cliente
05 order by nome
```

A linha 3 faz uso da função LEAD, que recebe como parâmetro o campo que será retornado. O detalhe é que o valor é o do próximo registro em relação ao que está sendo exibido, como mostra a **Figura 8.**

Existem outras funções da categoria navegacional. São elas FIRST_VALUE e LAST_VALUE, que trazem respectivamente o primeiro e último valor de um determinado campo de uma ordenação. Temos também LAG, que retorna o valor anterior ao que está sendo impresso. A **Listagem 15** exemplifica a utilização de LAG em conjunto com LEAD, enquanto que a **Figura 9** exibe seu resultado final.

Utilização de expressões regulares

O Firebird 3 permite o uso de expressões regulares em consultas e no uso de constrainsts através do comando SIMILAR TO ESCAPE. Mas o que é uma expressão regular?

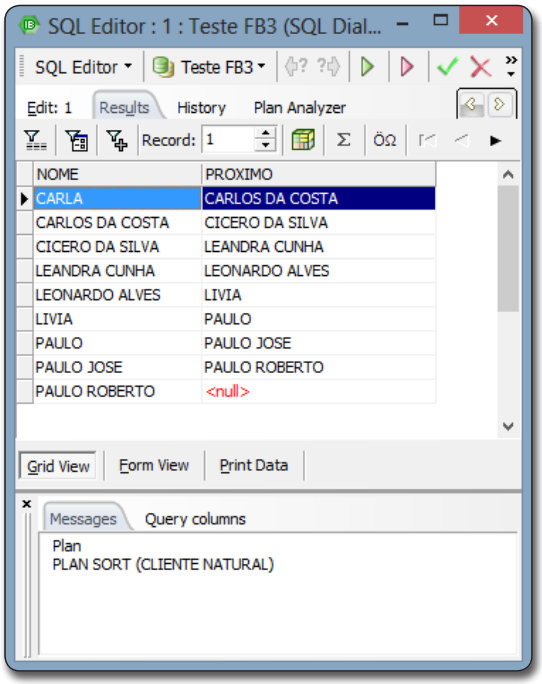

Figura 8. Uso da função LEAD

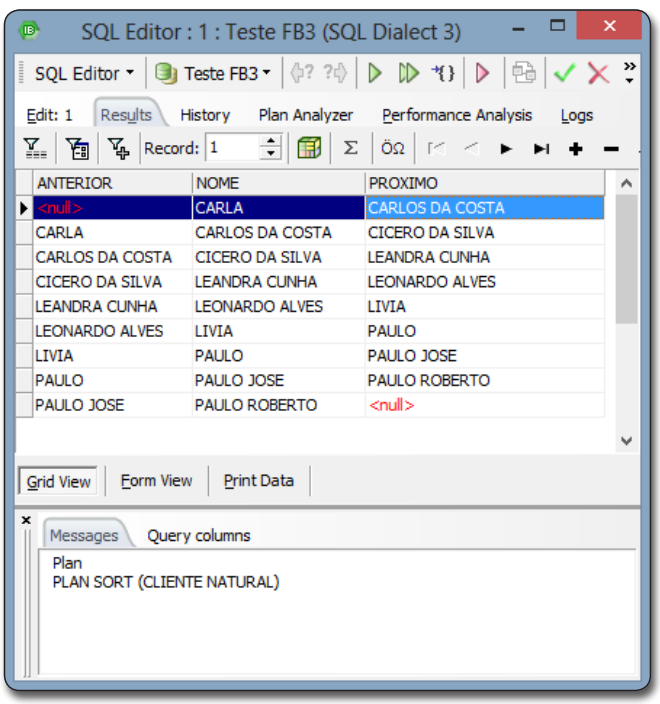

Figura 9. Resultado da **Listagem 15**

Listagem 15. Utilizando funções de navegação

```
01 select
02   lag(nome) over (order by nome) anterior,
03   nome,
04   lead(nome) over (order by nome) proximo
05 from cliente
06 order by nome
```

Uma expressão regular, também chamada de REGEX, é uma ferramenta de pesquisa e substituição de textos muito sofisticada. Usualmente, uma expressão regular é uma composição de caracteres e símbolos chamados meta-caracteres, cuja principal função é pesquisar um padrão conforme as condições fornecidas; nesse caso, diz-se que o texto encontrado "casou" com a expressão regular.

Muitas vezes, as expressões regulares são confundidas com caracteres curingas que normalmente utilizamos em pesquisas de arquivos, como por exemplo, "*.doc". Na verdade, as expressões regulares também são capazes de realizar pesquisas desse tipo, no entanto, elas são imensamente mais poderosas do que isso, permitindo escolher se a expressão a ser pesquisada está no começo ou no final da linha, quais os caracteres são permitidos, quantas vezes a expressão deve se repetir, entre outros tipos de pesquisas.

O tema "Expressões regulares" é tratado com certo receio e, às vezes, deixado um pouco de lado pelos desenvolvedores Delphi, em virtude de que poucas pessoas conhecem o seu real potencial.

Além disso, ao nos depararmos pela primeira vez com as expressões regulares, a primeira reação é de rejeição e desconfiança; afinal, quem é capaz de entender essas expressões "esquisitas", como podem ser vistas no código a seguir:

```
^[a-z]{1,8}\.[a-z]{1,3}$
[_a-zA-Z\d\-\.]+@([_a-zA-Z\d\-]+(\.[_a-zA-Z\d\-]+)+)
(FTP|HTTP)://([_a-z\d\-]+(\.[_a-z\d\-]+)+)
```

As expressões regulares são formadas por meta caracteres, que desempenham um papel específico dentro de cada expressão. A seguir temos um resumos dos meta caracteres.

- **Ponto:** O ponto (.) é um meta-caractere curinga, normalmente utilizado em expressões quando não se sabe exatamente qual o caractere que se quer pesquisar. Em resumo, o ponto casa com qualquer caractere existente, imprimível ou não, inclusive com o caractere literal ponto. Mas, no entanto, devemos lembrar que, apesar de casar com qualquer caractere, o ponto é obrigatório e deverá existir no texto a ser pesquisado, caso contrário, a expressão regular não casará. Exemplo: Expressão bo.o casa com bozo, bolo, boto;
- **Asterisco:** O asterisco (*) é um tipo de caractere curinga que casa com zero, uma ou muitas ocorrências do caractere anterior a ele. Exemplo: Expressão 6*0 casa com 0, 60, 666666660;
- **Lista:** A lista ([]) indica classes de caracteres que devem existir no texto a ser pesquisado, a fim de que a expressão regular case. Além disso, podemos pesquisar classes de caracteres utilizando o caractere hífen para separar a faixa de caracteres que serão pesquisados (como por exemplo: 0-9, a-z, A-Z, entre outras). Exemplo: Expressão n[ãa]o casa com não, nao;
- **Lista Negada:** A lista negada ([^]) tem o mesmo comportamento da lista, porém indica as classes de caracteres que não devem existir no texto a ser pesquisado a fim de que a expressão regular case. Exemplo: Expressão [^a-z] casa com 1, ?, #;

• **Mais:** O mais (+) tem um comportamento semelhante ao asterisco. A única diferença é que o mais não é opcional, nesse caso, o caractere anterior ao mais deve casar pelo menos uma vez. Exemplo: Expressão bi+p casa com bip, biip, biiip;
• **Opcional:** O opcional (?) é um meta-caractere curinga semelhante ao ponto. A principal diferença entre o opcional e o ponto, é que o caractere anterior ao opcional não é obrigatório, ou seja, pode ter ou não ter a ocorrência desse caractere para que a expressão regular case. Além disso, o opcional também é capaz de tornar outras regras definidas por outros meta-caracteres opcionais. Exemplo: bol?a casa com bola, boa;
• **Escape:** O escape (\) é o meta-caractere utilizado quando precisamos pesquisar um caractere que é igual a um dos meta-caracteres utilizados pelas expressões regulares. Nesse caso, se precisarmos pesquisar um caractere ponto, temos que utilizar o escape para informar ao motor de expressões regulares que aquele ponto que está descrito na expressão regular não é o meta-caractere ponto, é na verdade o caractere que deve ser procurado. Exemplo: \. casa com . ;
• **Circunflexo:** O circunflexo (^) é uma espécie de "âncora" que indica os caracteres que devem estar no começo da linha ou texto a ser pesquisado, para que a expressão regular case. Exemplo: ^Delphi casa com Delphi;
• **Cifrão:** O cifrão ($) é uma espécie de âncora semelhante ao circunflexo. Porém, a sua função é indicar os caracteres que devem estar no final da linha ou texto a ser pesquisado, para que a expressão regular case.
• **Ou:** O meta-caractere ou (|) é utilizado para criar expressões regulares com mais de uma alternativa possível. Exemplo: http://|ftp:// casa com http://, ftp://;
• **Grupo:** O meta-caractere grupo (()) tem a função de agrupar caracteres e meta-caracteres em blocos de expressões regulares. Além da criação de blocos de expressões, o grupo tem a função de ampliar a utilização dos outros meta-caracteres. Cabe lembrar que grupos podem conter outros grupos dentro dele. Exemplo: Delphi (XE|XE5) casa com Delphi XE, Delphi XE5;
• **Chaves:** O meta-caractere chaves ({m,n}) tem a função de determinar exatamente a quantidade de vezes que um texto deve repetir para que a expressão regular case. Exemplo: macar{1,2} ao casa com macarrao, macarao.

Para mostrar seu uso o primeiro registro da tabela produto teve retirado de seu campo descrição o caractere numérico.

CURSOS ONLINE

A Revista SQL Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

Feita para Desenvolvedores de Software e DBAs
SQL magazine

CONHEÇA OS CURSOS MAIS RECENTES:

- Cursos: Curso de noSQL (Redis) com Java
- Desenvolvimento para SQL Server com .NET
- Curso PostgreSQL - Treinamento de banco de dados (Curso Básico)

Para mais informações :
http://www.devmedia.com.br/cursos/banco-de-dados
(21) 3382-5038

DEVMEDIA

Assim, para localizar todos os produtos que não terminem com algum caractere numérico, usamos a instrução SQL da **Listagem 16**.

Ao observarmos os meta caracteres utilizados, constatamos que definimos uma lista negada de caracteres numéricos, e essa lista foi utilizada em conjunto com o caractere '%', que permite a busca em uma string de qualquer tamanho. Já o ESCAPE, que é obrigatório, está apenas definindo o "\" como caractere para identificação dos caracteres escape. As expressões regulares também podem ser utilizadas para localização de textos mal formatados, como e-mails ou telefones.

Listagem 16. Utilização de expressões regulares

```
01 select * from produto
02 where
03 descricao similar to '%[^0-9]' escape '\'
```

Definindo a segurança

Até o momento utilizamos o Firebird 3 em sua versão local, que dispensa um usuário e senha. Para realizarmos uma conexão através de uma conexão de rede (localhost, por exemplo) é necessário criar um usuário e senha. Para realizar essa operação, é necessário primeiramente parar a execução do Firebird e abrir seu arquivo firebird.conf em um editor de texto simples. Com o arquivo aberto localizamos a configuração:

```
# SecurityDatabase = $(root)/security3.fdb
```

Ela é utilizada para determinar qual é o banco de dados de segurança, nos mesmos moldes da versão 2.X. Para ativar esse comportamento basta retirarmos o "#" e salvarmos o arquivo. Para a criação do usuário, ainda com a instância do Firebird desativada, acessamos sua pasta pelo prompt de comando e executamos o seguinte comando:

```
gsec –add paulo –pw Paulo –admin yes
```

Esse comando cria, com privilégios de administrador um usuário "paulo", de senha "Paulo" que poderá então ser utilizado para realizar uma conexão remota, através de rede. É preciso então reiniciar o Firebird 3, através do comando:

```
Firebird –a -m
```

A **Figura 10** mostra como fica a configuração de conexão via IBExpert.

A propriedade Server é marcada como Remote e em Server name definimos localhost/3053, indicando que estamos nos conectando a um servidor Firebird, localizado na mesma máquina, mas que escuta na porta 3053. Ajustamos as propriedades Username e Password para o usuário criado e confirmamos.

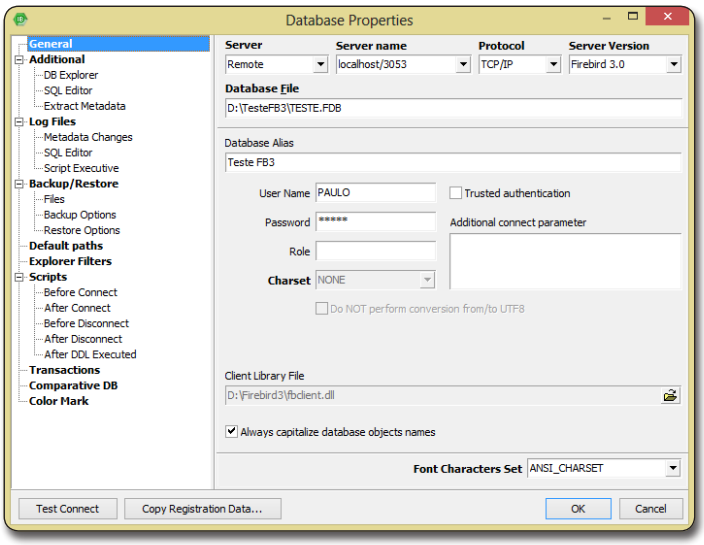

Figura 10. Acessando o Firebird via rede

Conclusão

O Firebird 3 traz novos recursos, alguns mostrados aqui foram implementados pelo integrante brasileiro, o Adriano dos Santos Fernandes, mas existem outros que não foram abordados. Apesar disso, uma coisa é certa, a migração para essa nova versão vai implicar em ajustes nas bibliotecas de acesso a dados e até mesmo uma revisão das sentenças SQL geradas pelos sistemas. Por exemplo, todo sistema possui "aquela" sentença SQL que está no aplicativo ou foi gerada a partir de uma Stored Procedure justificando um recursos que a sintaxe padrão não oferecia. Agora, temos REGEX nas consultas, isso traz uma grande flexibilidade, campos booleanos então foram muito bem vindos, sem contar o recurso de Identity.

Espero que este artigo tenha chamado a atenção para essa versão e despertado o interesse em aplicação de testes, pois o que mais a comunidade precisa nessa hora, é de pessoas dispostas a realizar um teste, a migrar uma base na versão 2.X para a 3 e ver como tudo se comporta e em caso de dúvida, relatar em seu respectivo site.

Autor

Paulo Quicoli
pauloquicoli@gmail.com
Editor da revista ClubeDelphi. Formado em processamento de dados pela FATEC-TQ. Atua como analista programador de sistemas da Siplan Control-M (www.siplancontrolm.com.br) unidade Jaboticabal, e professor da FATEC, campus Taquaritinga.
Blog: http://quicoli.worpress.com

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Desenvolvimento Android no Delphi

Construindo aplicações para Android com o novo Delphi XE5

O suporte ao desenvolvimento móvel no Delphi surgiu ainda em sua versão XE2, até então direcionado exclusivamente à construção de aplicativos para a plataforma iOS, contemplando dispositivos Apple, tais como iPhone, iPad e iPodTouch. Adicionalmente, ainda nesta época já se ventilava os primeiros rumores sobre o eventual suporte nativo da ferramenta ao desenvolvimento móvel para Android, num futuro próximo. Isto porque, apesar da clara e reconhecida evolução para a época, o suporte a iOS atendia apenas parcialmente o nicho de mercado ao qual a grande maioria dos desenvolvedores Delphi estavam inseridos. Sua realidade Mobile envolvia, além dos dispositivos iOS, um grande mercado em expansão e abrangente de dispositivos Android, vide smartphones e tablets dos mais variados tipos e marcas. Logo, surgiram os primeiros pedidos sobre o mencionado suporte e, passadas algumas versões da ferramenta, tal promessa veio a ser finalmente atendida com o recém-lançado Delphi XE5. Sendo assim, esta versão da ferramenta conta com o novo suporte nativo ao desenvolvimento de aplicações para a plataforma Android. Além disso, o grande diferencial neste ponto fica por conta da forma de desenvolvimento aplicada à construção deste novo tipo de aplicação, que se mostra essencialmente a mesma da já utilizada para o desenvolvimento clássico para Win32. Em outras palavras, isso reflete o envolvimento de uma mesma linguagem, uma mesma sintaxe e os mesmos componentes já conhecidos. Ao final das contas, todo este cenário vem ilustrar a pretensão maior da Embarcadero, que é a de prover a construção de aplicações para múltiplas plataformas (Windows, Mac OS X, iOS e Android), atendendo múltiplos dispositivos (PCs, Smartphones e Tablets) a partir de uma mesma base de código e de conhecimento.

De forma natural, o desenvolvimento de aplicações móveis por si só envolve uma série de fatores extras que extrapolam os recursos providos internamente pela ferramenta de desenvolvimento. Com o Delphi não é diferente e, sendo assim, a pretensão deste artigo é a de justamente prover o maior número de informações e detalhes que norteiam o início do desenvolvimento para Android no Delphi, servindo assim como ponto de partida para a construção das primeiras aplicações deste tipo.

Fique por dentro

A mais nova versão do Delphi, a XE5, apresentou como grande atrativo o suporte nativo ao desenvolvimento móvel para Android. Esta novidade veio a atender um desejo antigo da comunidade, que enxerga a referida plataforma como sendo uma ótima opção comercial. Isso pode ser justificado pelo fato do Android ser atualmente o SO mais difundido e utilizado em meio ao contexto dos dispositivos móveis (tablets e smartphones). Em vista disso, o presente artigo irá mostrar os detalhes iniciais que envolvem a construção de aplicações deste tipo, através do uso do IDE do Delphi XE5.

Preparação do ambiente

Logo de início é possível estabelecer que o desenvolvimento para Android no XE5 requer uma preparação pontual do ambiente de trabalho. Diferente do que acontecia em sua abordagem mais tradicional, de desenvolvimento de aplicações para Windows, este novo processo construtivo no Delphi exige uma série de fatores "extras", que vão além dos recursos providos internamente pelo IDE. Em vista desse diferencial, a seguir são discorridos cada um dos itens necessários ao desenvolvimento para Android nesta nova versão do Delphi.

Android Tools

Dentre os recursos externos ao IDE do Delphi, necessários ao desenvolvimento de aplicações para a Android, três são tidos como fundamentais: JDK, NDK e SDK. Tais siglas representam elementos os quais podem ser traduzidos como sendo ferramentas de desenvolvimento para a referida plataforma, essenciais a

qualquer ambiente voltado à criação de aplicações móveis deste segmento. Os dois primeiros podem ser considerados recursos estáticos, uma vez que a instalação/atualização de ambos se dá com pouca frequência no ambiente de desenvolvimento. Num processo oposto, o SDK pode ser considerado variável a cada versão da plataforma.

De início, a presença do JDK (Java Development Kit) é justificada pelo fato do Android estar diretamente atrelado à plataforma Java, tornando-se necessária a presença de seu kit de desenvolvimento oficial, que habilita a criação de softwares para esta tecnologia. No contexto do Delphi XE5, a versão requerida e recomendada do JDK é a 1.6 ou 1.7. Em vista disso, a fim de prevenir eventuais transtornos, caso o ambiente não contemple nenhuma dessas versões, a própria instalação do XE5 se encarrega de instalar o JDK adequado.

Já NDK (Android NDK) é o acrônimo para Native Development Kit e pode ser brevemente definido como sendo um conjunto de recursos que permite a implementação de partes da aplicação usando linguagens de código nativo, tais como C e C++. Sua finalidade de uso destina-se essencialmente à geração de aplicações autossuficientes e que façam uso direto e intensivo do processador do dispositivo (CPU), sendo então muito útil à mecânica interna utilizada nos bastidores do Delphi. Em outras palavras, através do uso do NDK é que o Delphi gera o código nativo necessário para a elaboração de suas aplicações móveis para a plataforma Android. Logo, a própria instalação da ferramenta contempla o instalador da versão adequada (e recomendada) do NDK. Por fim, SDK (Android SDK) remete ao termo em inglês Software Development Kit que, neste cenário, caracteriza o kit de desenvolvimento para a plataforma. Em suma, este pacote fornece as bibliotecas e ferramentas necessárias a todo o contexto que envolve a construção de aplicativos para Android. Apenas como informativo, a seguir é exposto o caminho padrão de cada uma das ferramentas, tendo como base uma instalação padrão do release de lançamento do Delphi XE5:

JDK – C:\Program Files\Java\jdk1.7.0_25;
NDK – C:\Users\Public\Documents\RAD Studio\12.0\PlatformSDKs\android-ndk-r8e;
SDK – C:\Users\Public\Documents\RAD Studio\12.0\PlatformSDKs\adt-bundle-windows-x86-20130522\sdk

Em complemento a este cenário, o Delphi XE5 conta com um elemento externo denominado Android Tools, que fica disponível através do menu de instalação da ferramenta, a partir da guia Iniciar do Windows, conforme mostra a **Figura 1**. Uma vez acionada tal opção, imediatamente é aberta a janela Android SDK Manager (**Figura 2**), que apresenta um prognóstico geral dos recursos disponíveis a cada versão da plataforma, mostrando o status de cada item (se instalado ou não) e possibilitando inclusive a eventual instalação ou atualização de cada um. Assim como acontece com os itens anteriores (JDK e NDK), o próprio instalador do Delphi XE5 já provê, de forma opcional, a instalação de uma versão adequada do SDK (**BOX 1**). Tal facilidade é oferecida logo no início do processo, tal qual mostrado na **Figura 3**.

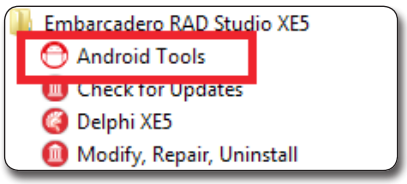

Figura 1. Android Tools

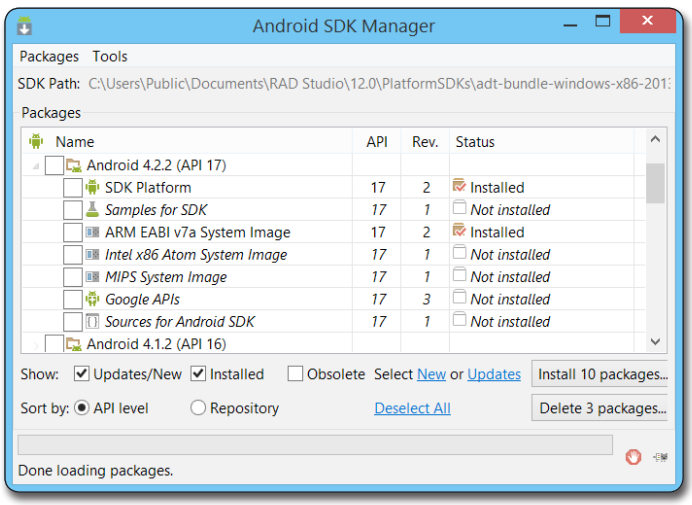

Figura 2. Android SDK Manager

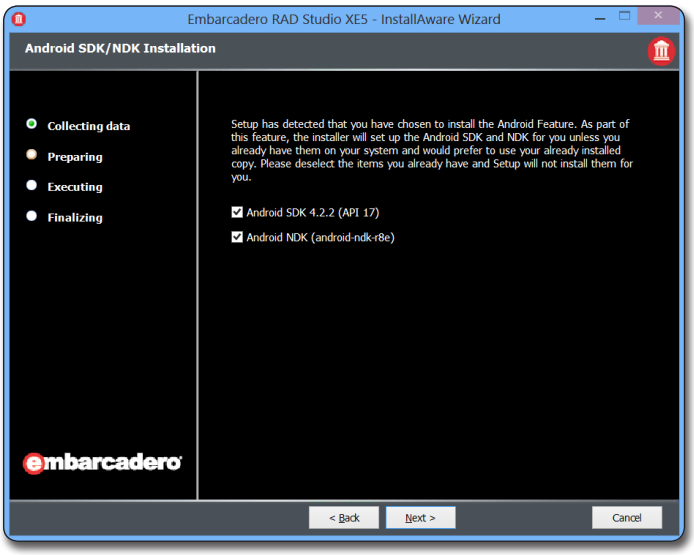

Figura 3. Instalação do Delphi XE5 – Android SDK e Android NDK

Em complemento, para que o desenvolvimento Android seja efetivamente habilitado no IDE, é necessário que tanto NDK quanto SDK sejam identificados internamente. Para tal, agora o próprio menu de opções da ferramenta (menu Tools | Options) conta com um novo elemento também denominado SDK Manager,

que traz uma lista dos caminhos condizentes a cada propriedade relacionada à versão do SDK utilizado, incluindo o NDK. De posse destas informações é que o Delphi consegue detectar a localização das bibliotecas a serem utilizadas durante o processo de desenvolvimento. Conforme pode ser visto na **Figura 4**, uma vez que é feita a opção pela instalação da versão do NDK e do SDK providos nativamente, as propriedades de ambos já são automaticamente detectadas pelo IDE.

Pelo fato de que cada versão da plataforma está atrelada a um SDK específico, é natural que o desenvolvedor venha a fazer uso de um SDK diferente do provido nativamente. Em situações mais tradicionais, é comum que o ambiente já conte com uma instalação existente de SDK. Para estes casos, torna-se necessária a adição manual do Android SDK em questão, ao SDK Manager do IDE.

BOX 1. Android SDK e Android NDK – Versões recomendadas

A instalação do Delphi XE5 contempla, de forma nativa e opcional, uma versão recomendada do Android SDK (4.2.2) e do Android NDK (android-ndk-r8e). Aqui, se diz recomendada (ou adequada) pelo fato destas versões serem aquelas utilizadas pela equipe de desenvolvimento da Embarcadero durante a elaboração do produto, tendo assim a garantia de serem plenamente suportadas pelo IDE. Por conseguinte, a utilização de versões diferentes destas pode implicar na existência de nuances não plenamente suportadas pela versão corrente do IDE, ocasionando comportamentos indesejados.

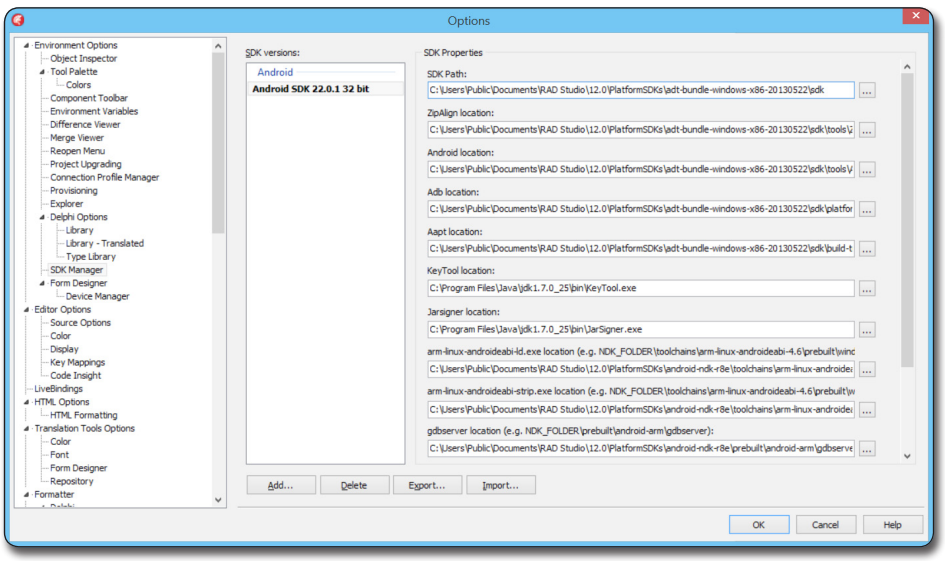

Figura 4. Opções IDE – SDK Manager

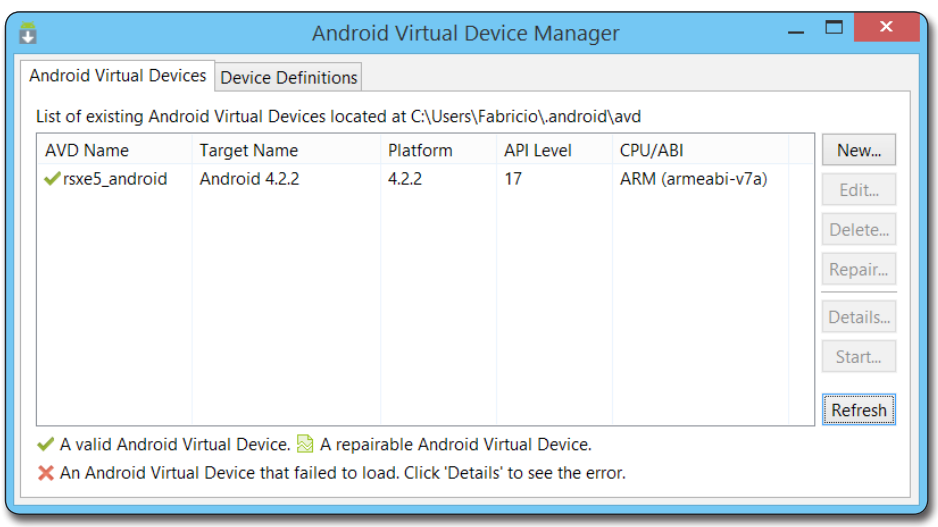

Figura 5. Android Virtual Device Manager

USB Driver

No IDE do Delphi, a execução de uma aplicação Android se dá basicamente de duas formas, sendo através do envolvimento de um software emulador ou de um dispositivo físico real. Para este último, o processo envolve a simples conexão do aparelho ao hardware de desenvolvimento, que ocorre via porta USB. Diante disso, é possível determinar que outro requisito fundamental para o desenvolvimento Android no XE5 está relacionado ao uso do driver USB apropriado ao dispositivo envolvido. Como exemplo, para um aparelho Nexus 7, se torna necessário o uso do Google USB Driver, da mesma forma que um aparelho Motorola exige o driver USB específico, provido pela própria fabricante. Por padrão, dentro do contexto do Delphi XE5, grande parte das necessidades relacionadas à drivers, tais como instalação e atualização, podem ser sanadas através do uso de seu Android SDK Manager.

Emulador

Sem sombra de dúvidas, a forma mais natural de se executar uma aplicação Android no Delphi é através do uso de um emulador. Por esta razão, o próprio instalador do Delphi, por meio da instalação nativa do SDK e NDK do Android, já provê a criação de um emulador padrão já configurado para o devido uso no IDE, em meio ao processo de desenvolvimento de aplicações móveis para a plataforma. Este emulador padrão é apresentado sob o nome de rsxe5_android e sua definição completa, incluindo suas configurações, podem ser vistas através do gerenciador de dispositivos virtuais do Android, denominado Android Virtual Devices (**Figura 5**), provido pelo Android SDK Manager, e disponibilizado através de seu menu Tools | Manage AVDs.

FireMonkey Mobile Application

Em sua versão anterior, a XE4, o Delphi trouxe como uma de suas grandes novidades um novo tipo de projeto,

denominado FireMonkey Mobile Application que, conforme seu nome sugere, está direcionado à construção de aplicações móveis com a ferramenta. Nesta época, conforme já dito, o suporte do IDE ao contexto Mobile se restringia à plataforma iOS, resultando na produção de aplicativos para dispositivos como iPhone, iPad e iPod Touch. De forma natural, nesta sua nova versão (XE5), a ferramenta teve seu suporte aumentado com a inserção da plataforma Android a seu contexto. Como reflexo disso, agora a partir de um tipo de projeto único (FireMonkey Mobile Application), é possível se construir aplicativos para ambas as plataformas móveis suportadas. O direcionamento específico a cada plataforma se dá simplesmente pela devida indicação na opção Target Platforms do Project Manager do IDE. A **Figura 6** mostra as opções habilitadas de forma nativa em projetos deste tipo no XE5. Ainda com base na imagem, um detalhe a ser mencionado neste momento se dá pela indicação do Android, como sendo a plataforma de destino padrão para o tipo de projeto FireMonkey Mobile Application. Isso se deve muito ao fato do suporte a Android ser uma das grandes apostas da Embarcadero, nesta nova fase do Delphi.

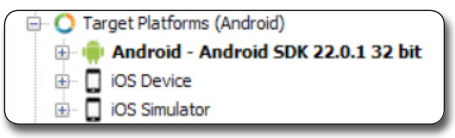

Figura 6. Project Manager – Target Platforms

Templates para projetos FireMonkey Mobile

Ao se iniciar um novo projeto FireMonkey Mobile Application no Delphi XE5, uma gama de modelos pré-definidos (templates) de aplicações são disponibilizados ao desenvolvedor, servindo como ponto de partida para o efetivo desenvolvimento. A função principal destes modelos previamente elaborados é facilitar o início do processo construtivo do aplicativo, evitando que se comece totalmente do "zero". Em suma, cada modelo conta essencialmente com formulários e componentes pré-configurados, de acordo com sua característica relacionada. Por esta razão, cada template disponibilizado é autodocumentado, o suficiente para que o desenvolvedor tenha a clara noção de sua intenção e característica. Tais informações são mostradas pelo simples repousar do ponteiro do mouse sobre cada modelo. A **Figura 7** mostra alguns dos templates Mobile disponíveis no XE5.

Com exceção do modelo Blank Application, cada template deste, quando utilizado, exige a indicação imediata de um local a ser salvo em disco, uma vez que não são gerados em memória, mas sim copiados fisicamente a partir do repositório do Delphi.

Um exemplo prático: primeira aplicação Android no Delphi XE5

Mantendo a tradição no que diz respeito à apresentação prática de uma nova abordagem relacionada a desenvolvimento de software, de início será construído um tradicional projeto "Hello World!", de forma a introduzir a construção de aplicações Android no Delphi. Sendo assim, utilizando como base o modelo de aplicação mais básico (Blank Application) de um projeto FireMonkey Mobile Application, a construção do layout a ser utilizado se dá de maneira similar ao já realizado ao longo dos anos com projetos VCL Forms Application.

Isto posto, pelo simples arrastar de um botão (TButton) para o formulário do projeto, fica definido o layout da aplicação (**Figura 8**). Ainda em termos visuais, outra alteração necessária se dá pela devida definição dos dizeres a serem mostrados no controle que, neste caso, seria algo como "Dizer Olá!". Numa abordagem tradicional utilizando a VCL, tal mudança implicaria na configuração da propriedade Caption do elemento TButton. Todavia, apesar das semelhanças notadas, tal como o nome da classe do controle utilizado (TButton), é importante lembrar que o desenvolvimento Mobile no Delphi está relacionado ao framework FireMonkey (FMX). Em função disso, os elementos utilizados tendem a ter suas próprias nuances. Especificamente para este caso, o detalhe fica por conta da propriedade do controle TButton que representa o texto a ser mostrado em seu visual.

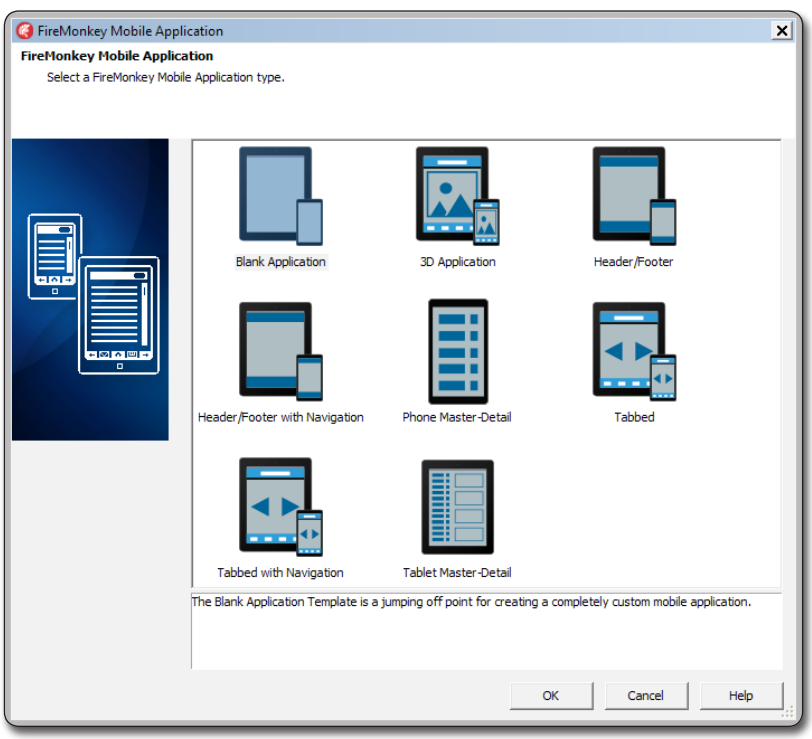

Figura 7. FireMonkey Mobile Application – Templates

Enquanto que na VCL é utilizada a propriedade Caption, aqui se utiliza a propriedade Text. Apesar de sutil, especificações como esta podem gerar confusões num desenvolvimento inicial.

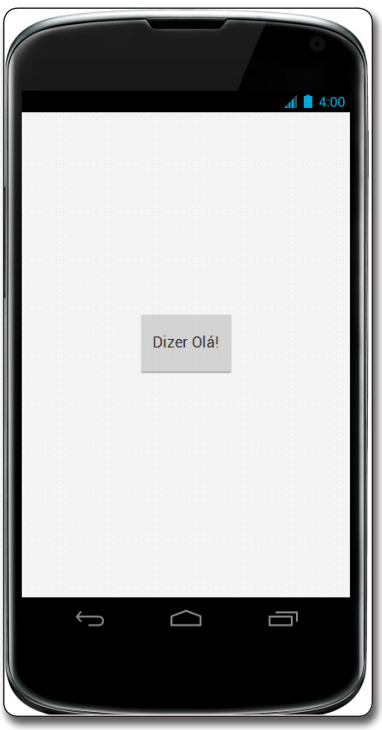

Figura 8. Layout da primeira aplicação de exemplo

Já em termos de codificação, a **Listagem 1** mostra o código do método manipulador do evento OnClick do botão. Da mesma forma que o habitual, apenas uma chamada ao método ShowMessage é o suficiente para se mostrar uma mensagem informativa ao usuário. O detalhe aqui fica nos bastidores, uma vez que o ShowMessage utilizado faz referência ao método declarado na unit FMX.Dialogs, em substituição a VCL.Dialogs.

Listagem 1. Código para o evento OnClick do botão

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Olá Mundo!');
end;
```

Por conseguinte, já com a aplicação em execução, o resultado da ação do clique no botão é ilustrado na **Figura 9**.

Aplicação Android com banco de dados

Desde seu início, o Delphi teve sua excelência refletida na construção de soluções para a plataforma Windows e, fundamentalmente, no provimento de aplicações envolvendo banco de dados, conhecidas em meio à comunidade como "aplicações de banco de dados". Dito isto, mediante esta sua nova fase evolutiva, que abrange a construção de aplicações para plataformas móveis (iOS e Android), torna-se interessante o uso de uma abordagem já familiar para a demonstração dos novos conceitos envolvidos, traçando um paralelo entre elas sempre que possível.

Figura 9. Aplicação "Hello World!" em execução

O desenvolvimento de aplicações com banco de dados no Delphi XE5 para plataformas Mobile, sob o ponto de vista nativo, está estritamente relacionado ao uso dos bancos de dados Interbase ToGo e IBLite (**BOX 2**). Contudo, já de início vale ressaltar que ambos os produtos são comerciais (pagos), logo, sua utilização requer a posse de uma licença válida. Com base no contexto do artigo, que está relacionado ao uso da versão experimental (Trial) da ferramenta, o desenvolvimento se dá pela utilização da versão ToGo, uma vez que uma licença do produto (também Trial) é provida junto da instalação do IDE.

De posse da tecnologia de banco de dados, se faz necessária à definição da biblioteca de componentes de acesso a dados a ser utilizada. Para aplicações Android, três são as opções nativas: dbExpress (**BOX 3**), InterBase Express (IBX) e FireDAC. Levando em conta o nível de popularidade, num momento inicial, o dbExpress se mostra uma opção agradável. Em se tratando de plataformas móveis (iOS e Android), o dbExpress tem seu suporte direto limitado a dois tipos de banco de dados: o já citado InterBase ToGo e o SQLite. Para a plena utilização dos demais SGBDs suportados pelo framework, um cenário ideal se daria pelo envolvimento de um servidor de aplicação DataSnap, caracterizando assim um contexto multicamadas, o que foge do escopo deste artigo.

Banco de dados

Ainda com relação ao InterBase ToGo, a própria instalação do XE5 traz consigo uma base de dados de exemplo desta versão do SGBD, cujo arquivo apresenta-se sob o nome de "DBDEMOS.GDB",

VOCÊ TAMBÉM ESTÁ BOIANDO NESTA SOPA?

e que fica localizado na pasta Data, dentro do diretório de exemplos (Samples) da ferramenta (C:\Users\Public\Documents\RAD Studio\12.0\Samples\Data, tomando como base uma instalação padrão). Todavia, vale ressaltar ainda que mesmo que tal arquivo apresente uma extensão (.gdb) que remeta às versões Server ou Desktop do InterBase, sua utilização, em termos práticos, é circunstancialmente diferente.

BOX 2. InterBase ToGo e IBLite

De forma sucinta, o InterBase pode ser definido como sendo um poderoso banco de dados relacional, de alto desempenho e segurança, pertencente à mesma empresa mantenedora do Delphi, a Embarcadero. A robustez de seus recursos o credenciam a estar no mesmo patamar de outras soluções já também consolidadas no mercado, tais como Oracle, MySQL, MS SQL Server, entre outros. Assim como seus concorrentes, o InterBase, como produto, contempla uma série de versões distintas, cada qual com suas especificidades. Dentre estas, as versão denominadas InterBase ToGo é aquela plenamente suportada pela plataforma Android. Isto porque, esta versão reflete uma engine de banco de dados que pode ser embutida diretamente na aplicação, sem a necessidade de qualquer tipo de instalação. Para o seu funcionamento, nenhum processo adicional é alocado na memória, se resumindo apenas ao uso da própria biblioteca do banco de dados.

BOX 3. dbExpress

De forma conceitual, o dbExpress pode ser simplesmente definido como sendo um framework de acesso à banco de dados presente no Delphi, cuja infraestrutura é totalmente escrita na própria linguagem utilizada na ferramenta. Através da utilização de drivers específicos, o dbExpress se torna apto a trabalhar com os principais SGBDs disponíveis no mercado, tais como Oracle, DB2, MS SQL Server, MySQL, InterBase, Firebird e SQLite. Historicamente, até a versão XE2 da ferramenta, é possível afirmar de forma unânime que o dbExpress ocupava o posto de principal opção nativa, em termos de acesso a dados no IDE. Com a chegada do XE3 e a posterior apresentação da biblioteca FireDAC, o framework ganhou então um concorrente à altura, sendo que atualmente ambos (dbExpress e FireDAC) dividem o posto de "principais opções nativas" para a parte de acesso a dados no desenvolvimento de uma aplicação Delphi.

De início, o primeiro ponto a ser levantado é com relação ao gerenciamento de um banco deste tipo. Tradicionalmente, uma base InterBase pode ser plenamente gerida por ferramentas como IBConsole e IBExpert, que internamente se utilizam da biblioteca cliente (client library) padrão do SGBD, denominada "gds32.dll", para a efetivação do processo. O uso, por padrão, desta biblioteca pode ser justificado pelo fato dela estar relacionada às versões mais clássicas do InterBase (Ex: Desktop e Server) e até então tidas como as mais usuais. Todavia, tecnicamente, a utilização de uma base de dados InterBase ToGo exige o uso de uma biblioteca cliente específica, denominada "ibtogo.dll", em substituição à "gds32.dll". Sendo assim, um ajuste essencial a ser feito se dá pelo simples redirecionamento da client library a ser utilizada pela conexão. Para fins de conhecimento, o arquivo "ibtogo.dll" é distribuído junto da instalação do XE5, ficando localizado dentro do diretório "Win32_to", disponibilizado por padrão em:

C:\Users\Public\Documents\InterBase\redist\InterBaseXE3

Ainda neste momento é importante ressaltar que ambas as bibliotecas cliente citadas (gds32.dll e ibtogo.dll) dispõem de uma mesma interface interna, possibilitando então que um mesmo arquivo de banco de dados (.gdb) se torne apto a atuar nas distintas versões do SGBD. Ciente disso, é possível então se construir uma base de dados para a versão ToGo da mesma forma habitual realizada com a versão Server. Sendo assim, a **Listagem 2** mostra o script para a criação de um banco de dados simples, para a utilização com a versão ToGo do InterBase.

Listagem 2. Script do banco de dados de exemplo

```
/*****CRIA O BANCO DE DADOS*****/
SET CLIENTLIB 'C:\Users\Public\Documents\InterBase\
  redist\InterBaseXE3\Win32_togo\ibtogo.dll';

CREATE DATABASE '[SeuCaminho]\DB_CLUBEDELPHI.gdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET WIN1252;

/*****CRIA A TABELA*****/
CREATE TABLE REVISTAS (
   ID_REVISTA  INTEGER NOT NULL,
   ARTIGO      VARCHAR(50),
   AUTOR       VARCHAR(50)
);

/*****DEFINE A CHAVE PRIMÁRIA*****/
ALTER TABLE REVISTAS ADD CONSTRAINT PK_REVISTAS PRIMARY KEY (ID_REVISTA);
```

Conforme pode ser visto, a listagem apresentada é simples e autoexplicativa o suficiente a ponto de não exigir nenhum comentário adicional.

Layout

Como já é de conhecimento, FireMonkey Mobile Application é o tipo de projeto a ser utilizado para a construção de uma aplicação Android no Delphi XE5. Além disso, mediante a essência da elaboração de uma aplicação simples, que somente lista um conjunto de informações de determinada tabela do banco de dados, o template Header/Footer torna-se um bom ponto de partida. Este modelo de aplicação traz como característica de layout, um formulário contendo barras de cabeçalho e rodapé, tal como mostra a **Figura 10**. Por conseguinte, vale salientar que essas barras não se tratam de novos componentes FireMonkey, mas sim de controles já habituais da VCL. Como exemplo, a barra superior, é definida por um simples TLabel, já todo estilizado para o contexto, com o devido alinhamento, cor, e fonte. Assim como ocorre com o controle TButton, a tradicional propriedade Caption não está disponível ao TLabel da FMX, sendo novamente substituída pela propriedade Text.

Em aplicações VCL, em cenários onde se deseja manipular ou simplesmente exibir dados provindos de um banco de dados, é

ASSIM, O MERCADO TE ENGOLE!

PARAR DE SE ATUALIZAR PODE SER FATAL!

O mercado muda a todo momento e você precisa estar atento sempre. Seja um assinante MVP e devore tudo o que há de mais relevante no mercado de TI.

Invista na sua carreira!

TENHA ACESSO A:

- +DE 260 CURSOS ONLINE
- 09 REVISTAS MENSAIS
- 7.850 VÍDEO-AULAS

POR APENAS 59,90 MENSAIS

ASSINATURA MVP
MOST VALUABLE PROFESSIONAL

QUEM TEM ESTÁ TRANQUILO.

DEVMEDIA

Acesse: www.devmedia.com.br/mvp

natural a utilização de controles Data-Aware, tradicionalmente tratados como "controles DB", tais como DBEdit, DBComboBox, DBGrid, entre outros. Diferente disso, é importante lembrar que o FireMonkey não contempla controles "DB", utilizando-se então de uma abordagem e uma mecânica diferente da VCL, no que diz respeito ao seu trabalho com dados. Sendo assim, tomando como base a situação do exemplo citado, que exige a listagem de dados a partir de uma tabela do banco, em projetos VCL Forms um controle do tipo TDBGrid poderia ser plenamente utilizado para a exibição dessas informações. Já em projetos FireMonkey Mobile, este controle poderia ser substituído por um simples TListBox, que tem como finalidade exibir um conjunto de itens em uma lista de rolagem.

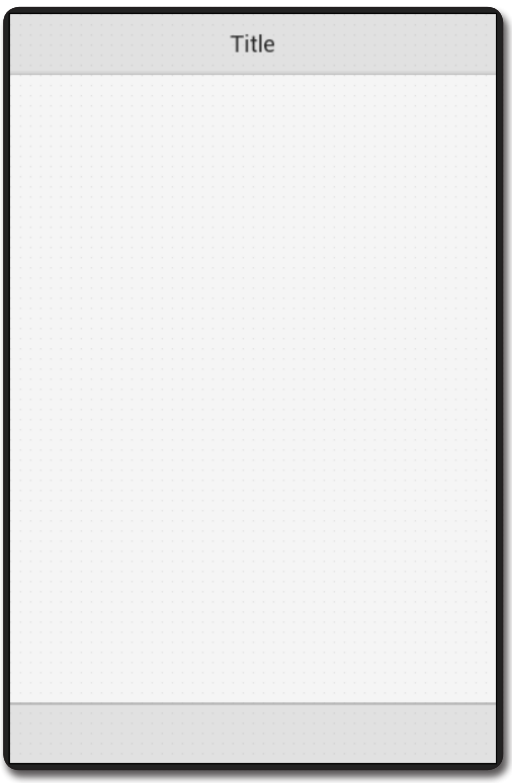

Figura 10. Header/Footer Template – Formulário

Em termos funcionais, este controle ainda exige a configuração de toda sua parte visual. Logo, uma primeira atitude a ser tomada diz respeito à estilização de sua caixa de listagem. Isso se dá através da subpropriedade ItemStyle, localizada em sua propriedade de nome DefaultItemStyles do próprio controle. Para um exemplo como o citado, uma sugestão é pela escolha da opção ListBoxItemBottomDetail, que determina que logo abaixo de cada item listado, seja mostrada uma pequena área para a exibição de seus detalhes.

O próprio controle provê ainda a possibilidade de outros ajustes adicionais, tal como a inclusão de uma caixa de pesquisa de itens, ao topo da listagem. Isto é conseguido por meio da inclusão de novos componentes relacionados, ao seu design, através de seu menu de contexto Add Item, conforme mostra a **Figura 11**. Pela imagem, é possível ver então a inclusão de um novo componente, do tipo TSearchBox.

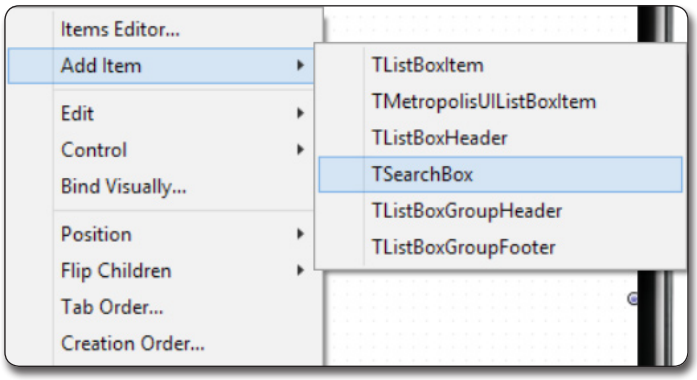

Figura 11. TListBox – TSearchBox

Acesso a dados

É possível estabelecer que a essência do exemplo citado esteja relacionada à sua parte de acesso aos dados da tabela do banco de dados. Uma vez que o dbExpress foi definido como sendo a tecnologia a ser utilizada para este fim, toda a sequente abordagem se dá de uma maneira muito próxima ao que é feito em projetos VCL. Como exemplo, TSQLConnection é o elemento central de conexão, contendo essencialmente as definições de driver (DBX) e de parâmetros da mesma. Para o primeiro, em razão do SGBD (InterBase ToGo) envolvido, IBLite/ToGo é o nome do driver a ser escolhido na propriedade homônima do componente. Adicionalmente, os parâmetros da conexão ficam definidos na propriedade Params, tal como mostrado na **Listagem 3**.

Listagem 3. Código DFM de um componente TSQLConnection – propriedade Params

```
object SQLConnection1: TSQLConnection
  ...
  Params.Strings = (
   'DriverUnit=Data.DBXInterBase'
   'VendorLib=ibtogo.dll'
   'VendorLibWin64=ibtogo64.dll'
   'VendorLibOsx=libibtogo.dylib'
   'Database=[CaminhoDoBanco]\DB_CLUBEDELPHI.GDB'
   'User_Name=sysdba'
   'Password=masterkey'
   'DisplayDriverName=IBLite/ToGo'
   ...
  )
end
```

O detalhe aqui fica por conta da exposição do uso da biblioteca cliente do InterBase ToGo (ibtogo.dll e ibtogo64.dll), bem como o caminho do banco, que deve ser adequado a um caminho local válido.

Uma vez de posse da conexão, o elemento que ficará responsável por fazer toda interação com a tabela desejada no contexto do dbExpress é o TSQLDataSet. Logo, sua configuração para este exemplo é mostrada na **Listagem 4**, onde fica clara a definição de uma simples instrução SQL em sua propriedade CommandText, que irá retornar os dados pertinentes aos campos da tabela REVISTAS.

Listagem 4. Consulta SQL

```
object SQLDataSet1: TSQLDataSet
  Active = True
  CommandText = 'select ID_REVISTA, ARTIGO,
   AUTOR from REVISTAS order by ARTIGO'
  SQLConnection = SQLConnection1
...
end
```

Por conseguinte, a finalização das configurações de tempo de design da aplicação envolve a ligação entre a parte de dados (TSQLDataSet) e seu visual, que aqui é representado pelo controle TListBox. Numa abordagem VCL, com a utilização de controles "DB", esta ligação se daria de forma natural através do uso de um simples TDataSource. Todavia, o trabalho com FireMonkey essencialmente se difere neste aspecto, uma vez que, conforme já citado, o framework não provê elementos "DB". A ligação aqui fica por conta do uso da tecnologia LiveBindings. Sendo assim, de forma prática, sua utilização, neste caso, se dá pelo uso do LiveBindins Designer, que é uma ferramenta visual disponibilizada através do menu View > LiveBindings Designer do IDE. Com sua estrutura aberta, este designer permite então que sejam definidas as ligações entre os elementos do formulário de uma maneira toda gráfica, tal como mostrado na **Figura 12**.

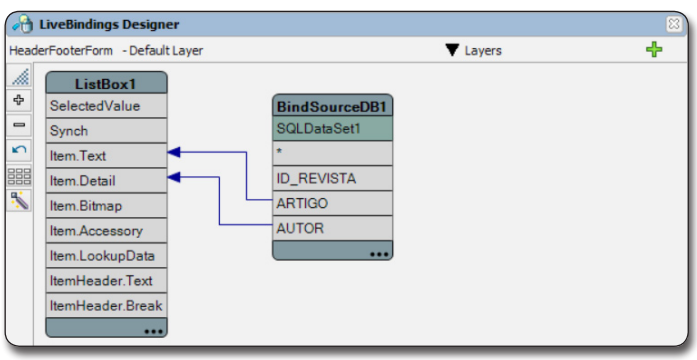

Figura 12. LiveBindings Designer

Fazendo o deploy da aplicação

Em situações reais, o passo seguinte ao desenvolvimento de uma aplicação está relacionado ao seu efetivo deploy para o ambiente real de produção. Tal processo pode ser brevemente definido como sendo uma coleta e posterior cópia para o ambiente de produção, de todos os arquivos necessários para a plena execução da aplicação. No caso de aplicações Mobile, o ambiente de produção é então o próprio dispositivo no qual a aplicação deverá rodar. Indo além, tecnicamente falando, o uso de um emulador em substituição a um dispositivo físico também exige o mesmo processo de deploy. No Delphi XE5, ao se executar uma aplicação em seu IDE, todo o processo de deploy é realizado de forma automática em seus bastidores.

A fim de prover um panorama visual dos arquivos envolvidos no processo, o próprio IDE do Delphi já provê uma solução rápida e ágil, disponibilizada em seu menu de projetos Project > Deployment. Uma vez acionada, esta opção traz à tona uma nova janela onde ficam listados os arquivos que irão fazer parte do deploy da aplicação. Por padrão, grande parte dos arquivos essenciais para que o aplicativo seja executado no dispositivo já são automaticamente inclusos nesta listagem.

Em razão do projeto de exemplo citado utilizar um recurso extra, que diz respeito ao banco de dados, se faz necessária a inclusão destes fatores na janela de Deployment. Em vista disso, uma primeira atitude é então incluir na lista de arquivos de deploy o arquivo do próprio banco de dados utilizado, por meio do botão de ação Add Files. Como ajuste, é necessário que seu caminho no dispositivo seja adequado, através da coluna Remote Path da grade de deployment. Em consequência desta mudança, se faz necessária também a adequação do caminho do banco, em runtime, do parâmetro da conexão relacionado (Database) no TSQLConnection. Isso pode ser feito de várias maneiras, de acordo com o Remote Path escolhido. A seguir, na **Listagem 5** são então mostradas duas alternativas tradicionais de uso na perspectiva Mobile, por meio do método manipulador do evento OnBeforeConnect do componente citado.

Listagem 5. Configurando o caminho do banco de dados

```
SQLConnection1.Params.Values['Database'] :=
  TPath.GetHomePath + PathDelim +
  NomeDoArquivoDeBancoDeDados;

SQLConnection1.Params.Values['Database'] :=
  TPath.Combine(TPath.GetDocumentsPath,
  NomeDoArquivoDeBancoDeDados);
```

Além disso, pelo fato do SGBD utilizado ser uma opção comercial (paga), ele necessita de uma licença válida para operar no dispositivo. Conforme já citado, a própria instalação da versão trial do Delphi XE5 já traz consigo uma licença habilitada do InterBase ToGo para estes fins. De modo geral, para se adicionar à lista de deploy todos os elementos relativos ao SGBD em questão, basta que se acione o botão de ação Add Featured Files. Uma vez feito isso, uma nova janela é aberta (**Figura 13**), já disponibilizando uma opção devidamente relacionada.

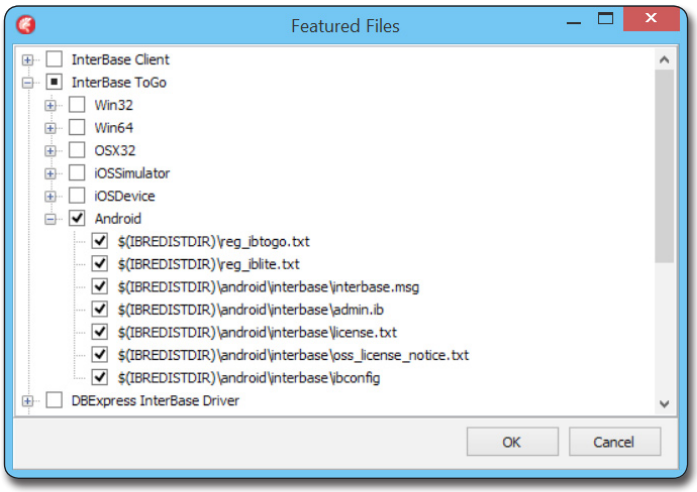

Figura 13. Deployment – Featured Files

Conclusão

A partir do pleno entendimento do cenário que envolve a construção de aplicações móveis para Android no novo Delphi XE5, o desenvolvedor pode tirar proveito de todas as vertentes que esta abordagem tem a oferecer. Vislumbrando situações hipotéticas, podemos citar desde a criação de uma simples aplicação móvel independente, que poderá ser distribuída de forma comercial em loja de aplicativos, passando por aplicações que envolvam dados armazenados no próprio aparelho em uma base local e, enfim, casos corporativos, onde a aplicação Android construída atua como um cliente de um Servidor de Aplicação DataSnap. Por conseguinte, são notórias a gama opções que são acrescidas ao contexto do Delphi, mediante seu novo suporte ao desenvolvimento Android.

Conforme já é de praxe no lançamento de cada nova versão, o Delphi XE5 acaba por proporcionar novos desafios aos membros de sua extensa comunidade. Portanto, novamente cabe ao desenvolvedor tirar proveito do que há de mais novo na ferramenta, ampliando ainda mais seu leque profissional.

Autor

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br

Formado em Processamento de Dados pela FATEC-TQ e pós-graduado em Engenharia de Componentes. Atua como Analista Programador Delphi há 8 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas. Certified Delphi Developer. Certified Embarcadero Instructor. Site pessoal: www.fabriciokawata.com.br

Links:

Delphi XE5 – Download Trial
https://downloads.embarcadero.com/free/delphi

Delphi XE5 – Página oficial do produto
http://www.embarcadero.com/br/products/delphi

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Seu **conhecimento** tem **visibilidade** na DevMedia

Publique artigos e vídeos no site e ganhe mais visibilidade em seu trabalho. Somos o maior site do Brasil para desenvolvedores.

Para publicar é bem simples: mande um email para nosso editor **Wesley Yamazack (yamazack@gmail.com)**.

Para mais informações acesse:
http://www.devmedia.com.br/seja-um-autor-devmedia/28103

DEVMEDIA

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.
Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486